Endfassung, 20. 2. 2000

GEOLOCATOR

Ein GPS-basiertes
Fahrzeugortungssystem

Diplomarbeit

Erstellt von:

Bernhard Liedl

Matrikelnummer: 8826842

Angefertigt am:

Institut für Informationsverarbeitung
und Mikroprozessortechnik (FIM)

Eingereicht bei:

o.Univ.Prof. Dr. Jörg Mühlbacher

COMMUNICATION & NAVIGATION

Abstract

The present paper documents the development of a vehicle tracking system, based on the NAVSTAR global positioning system. We will see that a tracking system consists of two basic tasks: handling gps receivers (or communication devices) on the one hand and displaying the tracks of located objects on a map on the other hand. These tasks will be done within separate dynamic link libraries, called GPSAPI (global positioning system's applications programming interface) and MAPAPI.

We will see that the combination of the global positioning system and the global system for mobile communication (GSM) enables a costeffective way for everyone, locating objects worldwide.

Furthermore, the document gives a summarized description of the NAVSTAR global positioning system as well as a general overiew of the global system for mobile communications (GSM).

Zusammenfassung

Die vorliegende Arbeit dokumentiert die Entwicklung eines auf dem NAVSTAR GPS basierenden Fahrzeugortungssystems. Zwei wesentliche Aufgaben sind in einem Fahrzeugortungssystem zu bewältigen: Die Handhabung von GPS-Empfängern (oder Kommunikationsgeräten) einerseits und die entsprechende Darstellung der von den Fahrzeugen empfangenen Daten auf einer Landkarte andererseits.

Die Kombination des Globalen Positionierungs Systems und des Global Systems For Mobile Communication (GSM) eröffnet eine kostengünstige Möglichkeit der weltweiten Fahrzeugortung für jedermann.

Darüberhinaus gibt diese Arbeit einen kurzen Überblick über das NAVSTAR Gobal Positioning System, sowie eine Übersicht über die Funktionsweise von GSM.

Danksagung

An dieser Stelle möchte ich Dank sagen: allen voran meinen Eltern, die mein Studium ideell und finanziell unterstützt haben. Mein ausdrücklicher Dank gilt auch dem geduldigen Betreuer meiner Diplomarbeit Herrn Dipl.Ing. Peter René Dietmüller.

Besonders danken möchte ich meiner Frau Gerlinde, daß sie mich in meinem Vorhaben immer wieder ermutigte und mir auch in schwierigen Zeiten eine liebevolle Stütze war, ebenso meinen Kindern Philip, Elisabeth, Sarah und David, die durch ihre fröhliche Art mich viele Dinge aus einer anderen Perspektive zu betrachten gelehrt haben.

Ottensheim, im Mai 1999 *Bernhard Liedl*

Inhalt

Einleitung

Vorwort

Im Laufe meiner Tätigkeit bei Communication & Navigation kristallisierten sich zwei Aufgabenstellungen heraus, die in (fast) jedem Software-Projekt im Bereich GPS-Datenerfassung zu bewältigen waren:

1. das Erfassen von GPS-Daten (damit verbunden die Steuerung von GPS-Empfängern verschiedener Hersteller, unter Umständen mit zwischengeschalteten Kommunikationsgeräten) und
2. die graphisch entsprechende, dynamische Darstellung von GPS-Positionen.

Es war daher naheliegend jeweils eine Bibliothek für beide Aufgaben zu entwickeln. Im Zuge meiner Diplomarbeit entstanden das *GPSAPI* (Global Positioning System Application Programming Interface) und das *MAPAPI* (Map Application Programming Interface).

Das GPSAPI sollte eine einheitliche Schnittstelle zur Steuerung von GPS-Empfängern unterschiedlicher Hersteller bieten, sowie GPS-Daten in einem herstellerunabhängigen Format zur Verfügung stellen.

Hauptziel des MAPAPI's war die Möglichkeit GPS-Daten dynamisch, also unmittelbar nach dem Einlesen darstellen zu können. Absicht war nicht ein Konkurrenzprodukt zu namhaften Herstellern von Geographischen Informationssystemen, wie etwa ESRI's MapObjects[1] zu entwickeln. Für Anwendungen, die sich rein auf die GPS-Datenerfassung konzentrieren, erwiesen sich große GIS-Systeme jedoch als überdimensioniert, und für den weniger erfahrenen Benutzer als nicht einfach zu bedienen.

Mit GPS allein läßt sich zwar die Positionen eines beliebigen Ortes bestimmen, für die *Ortung* eines Objektes benötigen wir jedoch noch ein geeignetes Medium zur Datenübertragung. Wir werden sehen, daß GSM eine für jedermann kostengünstige Möglichkeit darstellt, Daten weltweit drahtlos zu übertragen.

Nicht zuletzt muß hier auch die Frage nach der Verantwortung der Gesellschaft gegenüber gestellt werden. Sind wir George Orwell's *1984* ein Stück näher gekommen, oder stehen wir gar kurz davor? Einschlägige Anfragen bestätigen, dass man in Zukunft in diesem Zusammenhang wird wachsam sein müssen.

Zwei wesentliche Risiken und Gefahrenquellen sind:

- Das Rationalisierungs- und vor allem Kontrollpotential in der Arbeitswelt,
- sowie die Probleme des Datenschutzes und die damit verbundene Gefahr eines Überwachungsstaates.

[1] Environmental Systems Research Institute, Inc. gilt als einer der führenden Hersteller von GIS-Softwareprodukten (http://ww.esri.com)

Zielsetzung und Aufbau der Arbeit

Erstes Ziel der Arbeit war es, ein Application Programming Interface zur GPS-Datenerfassung zu entwickeln, in der Folge kurz *GPSAPI* genannt. Dieses API sollte mit GPS-Empfängern verschiedener Hersteller verwendbar sein. Demzufolge mußte dieses GPSAPI so konzipiert werden, daß es leicht um weitere GPS-Emfpänger erweiterbar ist, ohne daß bereits bestehende Komponenten des GPSAPIs neu compiliert werden müssen.

Ziel Nummer zwei war die Entwicklung eines MAPAPIs, eines Aplication Programming Interface zur visuellen Darstellung von GPS-Daten bzw. von Vektordaten.

Um diese beiden APIs herum sollte eine Anwendung entwickelt werden, die sich dazu eignet die beiden Komponenten ausgiebig zu testen aber auch die Leistungsfähigkeit der Produkte zu demonstrieren: der *GeoLocator*.

Kapitel 1 beschreibt das System auf das die Fahrzeugortung beruht: das Global Positiong System (NAVSTAR GPS). Die Komponenten des Systems werden vorgestellt, Möglichkeiten aber auch die Grenzen des Systems behandelt.

Kapitel 2 gibt eine Überblick über des *Global System For Mobile Communications,* besser bekannt als *GSM*. Es wird die grundsätzliche Funktionsweise des Systems erläutert, sowie die einzlenen Komponenten überblicksweise vorgestellt. Näher vorgestellt wird das *Short Message Service* (SMS), das, wie wir sehen werden, für die Fahrzeugortung von besonderer Bedeutung ist.

Kapitel 3 zeigt den GeoLocator als Instrument zur weltweiten Ortung von Objekten. Er stellt das Verbindungsglied zwischen GPSAPI und MAPAPI dar.

Kapitel 4 dokumentiert das GPSAPI als solches: Aufbau, Schnittstellen, Objektmodelle, interne Zusammenhänge, und die derzeit implementierten Gerätetreiber.

Kapitel 5 widmet sich dem API zur Darstellung von Kartenmaterial und GPS-Daten, dem Map Application Programming Interface.

Kapitel 6, die Benutzerdokumentation, zeigt wie der GeoLocator richtig konfiguriert wird und gibt einen Überblick darüber, wie der in der Praxis eingesetzt werden kann.

Anwendungsmöglichkeiten/Fallstudien

Die folgenden Anwendungsbeispiele sollen dazu dienen dem Leser Einblick in typische Aufgabenstellungen der GPS-Technologie zu geben. Es sind Beispiele, die in ähnlicher Form bereits verwirklich wurden, oder Ausschreibungsunterlagen entnommen sind.

Fallbeispiel 1

Die Landwirtschaftskammer läßt Bodenproben erheben. Eine manuelle Eintragung der Positionen in die DKM[2] ist zu ungenau (Abweichungen von mehr als 200 m) eine exakte Vermessung mittels konventioneller Vermessungstechnik wäre sehr zeitaufwendig und ist daher zu kostspielig.

Mit dem *GeoTracker™* (einem speziellen GPS-Empfänger der Firma Communication & Navigation) lassen sich nicht nur GPS-Positionen aufzeichnen sondern auch Daten eingeben. Damit läßt sich nicht nur die Position der Bodenprobe bestimmen, sondern es reduziert sich auch der Dokumentationsaufwand bei der Erhebung der Bodenproben selbst.

Zeitgleich zur Erhebung der Bodenproben zeichnet eine GPS-Referenzstation Korrekturdaten für ein später durchzuführendes Postprocessing auf. Nach der Erfassung im Feld werden die Daten vom GeoTracker mittels GPSAPI in den PC an *ArcGPS*[3] übertragen und mit Hilfe der parallel dazu aufgezeichneten Korrekturdaten korrigiert. Zur Kontrolle des Postprocessings kann man sich die Daten in *ArcGPS* auch graphisch darstellen lassen. Hier kommt das MAP-API zum Einsatz.

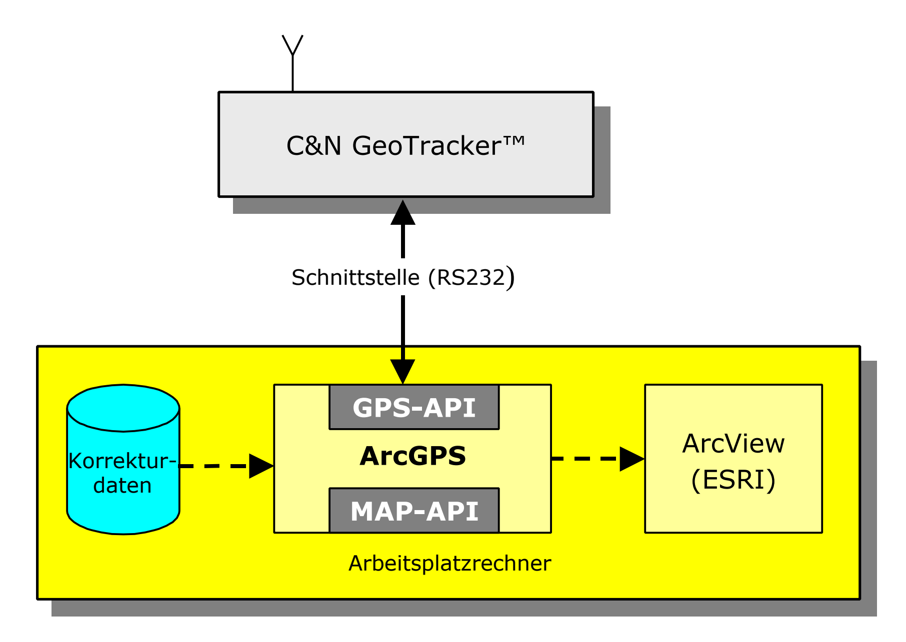


Abb. 1. GPS-API und MAP-API als Bestandteile von „ArcGPS"

[2] Digitale Katastralmappe
[3] *ArcGPS* ist ein Softwareprodukt von C&N das speziell für das Postprocessing von GPS-Daten und deren Export in Datenformate für Geographische Informations Systeme entwickelt wurde. Besonders wurde auf die Bedrüfnisse von ArcView-Anwender eingegangen, daher die Ähnlichkeit in der Namensgebung.

Fallbeispiel 2

In einem Speditionsunternehmen kommuniziert der Disponent mit den Fahrern der LKW-Flotte hauptsächlich über SMS. In unregelmäßigen Abständen benötigt er die Position bestimmter Fahrzeuge. Er sendet an das Fahrzeug mittels SMS eine Positionsanforderung. Der GPS-Empfänger beantwortet die Anforderung mit der aktuellen Fahrzeugposition. Die Fahrzeugposition soll in Map & Guide, das im Betrieb bereits zur Routenplanung eingesetzt wird, visualisiert werden.

Jedes Fahrzeug wird mit einem GPS-Empfänger und einem GSM-Modem (od. eingebautem GSM-Modul) ausgestattet. Der lokale PC wird ebenfalls mit einem GSM-Modem ausgestattet. Mithilfe des GPS-APIs ist es möglich sowohl Textnachrichten an den Fahrer, als auch Positionsanforderungen an den GPS-Empfänger im Fahrzeug zu senden.

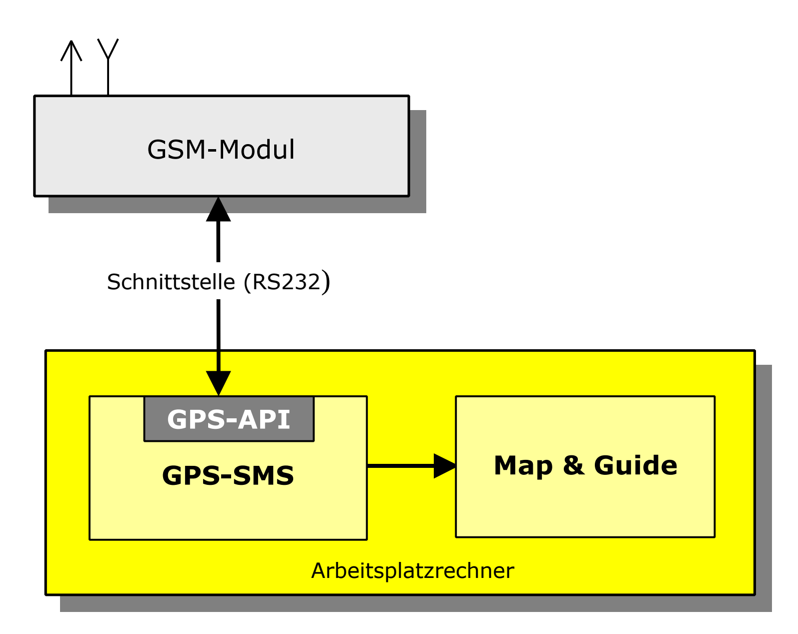


Abb. 2. Das GPS-API als Kommunikationstreiber für „GPS-SMS"

Fallbeispiel 3

Auf einem größeren Firmengelände soll aus Sicherheitsgründen die Position von Rangier-Lokomotiven, schweren Zugfahrzeugen und anderen Fahrzeugen laufend überwacht und mitprotokolliert werden. Es gibt mehrere Gruppen von Fahrzeugen, für die verschiedene Personen verantwortlich sind.

Die Positionsbestimmung erfolgt unter Einsatz der Differential GPS Technologie, um die geforderte Genauigkeitsansprüche von unter 5 m erfüllen zu können. Technische Grundlage für die Ermittlung der GPS-Korrekturdaten zur hochgenauen Positionsbestimmung der mobilen Einheiten ist die Verwendung einer GPS-Referenzstation an exponierter Lage.

Die Übertragung der Daten von den Fahrzeugen zu einer Zentrale erfolgt über einen eigenen Funkkanal des bereits vorhandenen, betriebsinternen Datenfunknetzes. Jedes der Fahrzeuge wird mit einem GPS-Empfänger und einem Funkmodem ausgestattet. Über ein geeignetes Protokoll übermittelt jedes der Fahrzeuge in periodischen Abständen (etwa alle 10 Sekunden) seine Position an die Funkstelle des zentralen Leitstandes.

Die Funkstelle des Leitstandes übermittelt die einkommenden Positionsdaten an den daran angeschlossenen „GPS-Server“.

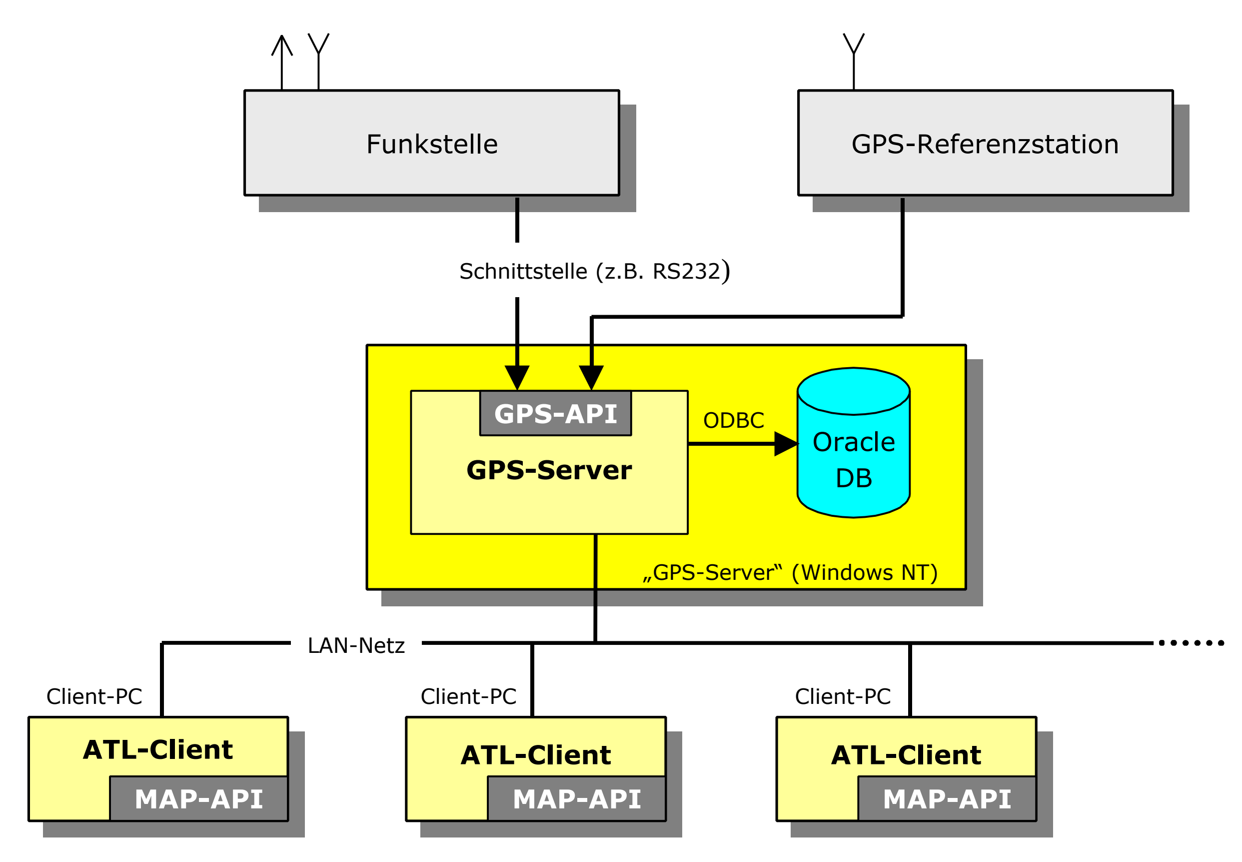


Abb. 3. Schematicher Aufbau des Client/Server Ortungssystems

Wesentliche Komponente der GPS-Server-Software ist das GPS-API, dessen Aufgabe es ist, die GPS-Daten und die Korrekturdaten der GPS-Referenzstation einzulesen, zu dekodieren und an das Hauptmodul weiterzuleiten. Der GPS-Server berechnet aus korrespondierenden GPS- und Korrekturdaten die exakten Positionen der mobilen Einheiten und legt diese in einer Oracle Datenbank ab. Zusätzlich werden die Positionsdaten an die Client-Anwendungen aktiv weitergeleitet.

Auf der Client-Seite wird das MAP-API zum Darstellen des Betriebsgeländes (Vektor-Daten oder Bitmap) und der Fahrzeugpositionen verwendet.

Kapitel 1:

GPS – Global Positioning System

NAVSTAR Global Positioning System

Der Krieg ist Vater aller Dinge.

Dieses Zitat trifft wohl auf kein anders System in höherem Ausmaß zu als auf Global Positioning Systeme. Wenn hier von GPS die Rede ist, so ist in erster Linie das vom amerikanischen Verteidigungsministerium in Auftrag gegebene und von diesem betriebene NAVSTAR Global Positioning System gemeint. Das ***NAV****igation* ***S****atellite* ***T****iming* ***A****nd* ***R****anging Global Positional System* ist ein wetterunabhängiges, weltweit verfügbares Satelliten-Navigationssystem. Neben seinem militärischen Einsatzbereich wurde aber auch sehr bald die Anwendungsmöglichkeit im zivilen Bereich erkannt.

Zielsetzung von GPS

Es sollte die Navigation (d.h. die Bestimmung von Position und Geschwindigkeit) eines beliebigen bewegten oder ruhenden Objektes ermöglichen. Darüberhinaus sollte auch noch eine genaue Zeitinformation zur Verfügung gestellt werden. Die Resultate sollten in Echtzeit, also ohne merkbaren Zeitverzug unmittelbar nach den Messungen, verfügbar sein. Weiters wurde die Forderung gestellt, bei jedem Wetter, zu jeder Zeit und an jedem beliebigen Ort auf oder nahe der Erde (also auf dem Land, auf dem Wasser und in der Luft) zu funktioieren.

Die Realisierung von NAVSTAR GPS

Alle bisherigen Navigationsverfahren sind nur in einem mehr oder weniger begrenzten Gebiet verfügbar, haben unterschiedliche Genauigkeiten und erlauben jedes für sich alleine keine dreidimensionale Positionsbestimmung. Sie sind teilweise sehr groß, unhandlich, schwer, teuer und „relativ" ungenau.

Es war klar, daß ein neues, allen bekannten Systemen überlegenes Navigationsverfahren nur vom Weltraum aus über Satelliten betrieben werden konnte. So wurde vom DOD das NAVSTAR-Satelliten-System und von der Sowjetunion das äquivalente GLONASS[4]-Satelliten-System entwickelt. Das Ergebnis war auf beiden Seiten ein Navigationssystem, das allen anderen bisher entwickelten Systemen in vielerlei Hinsicht überlegen ist. Es ist

---

[4] Abk. für Global Navigation Satellite System

bis in große Höhen weltweit verfügbar, ermöglicht eine ununterbrochene Versorgung zu jeder Tages- und Nachtzeit und dies bei jedem Wetter.

Das NAVSTAR GPS läßt sich in drei sog. „Segmente“ gliedern:

- *Raumsegment,*
- *Benutzersegment* und
- *Kontrollsegment.*

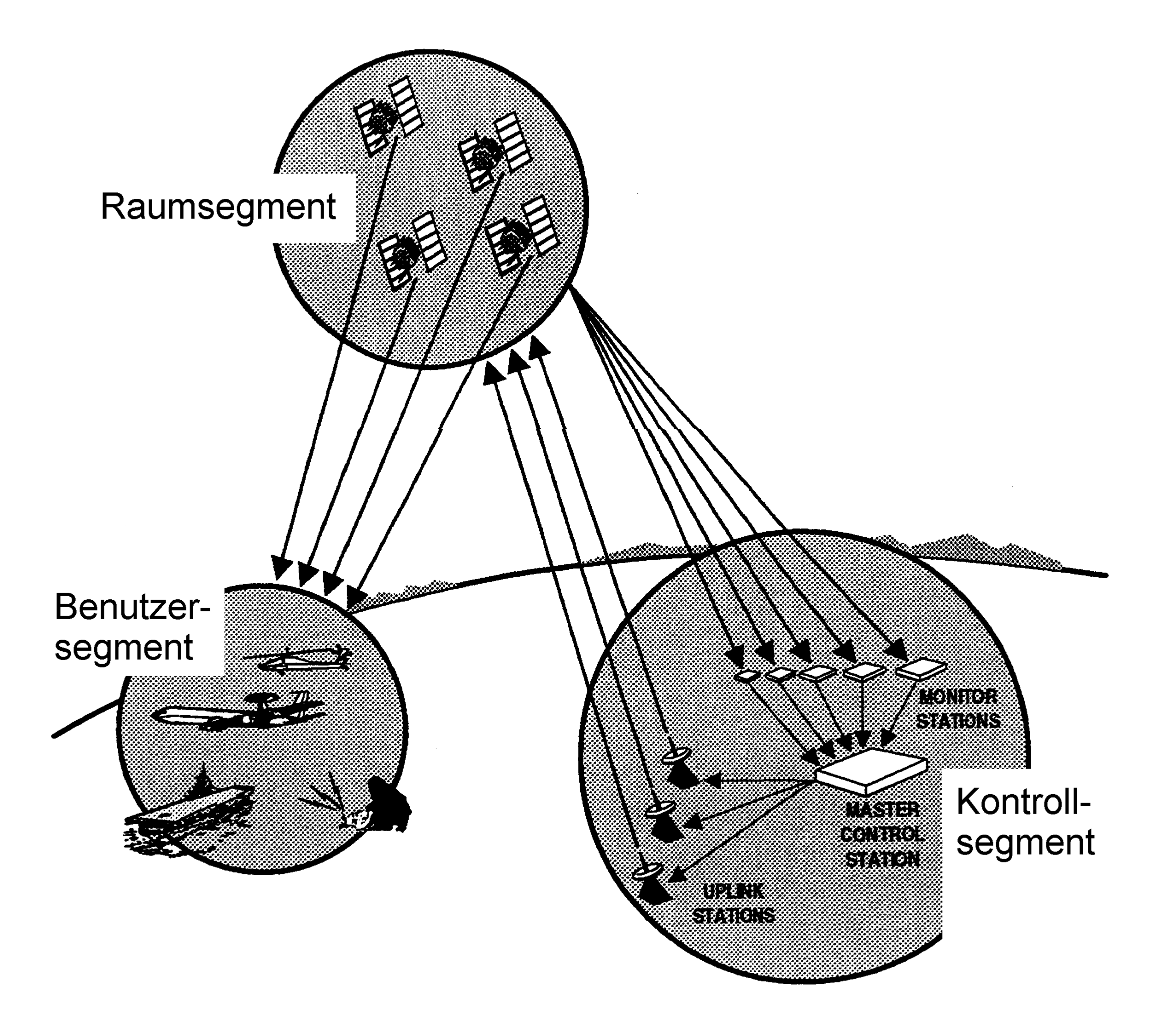


Abb. 4. Die Segmente von NAVSTAR GPS

Das Raumsegment

Das Raumsegment besteht aus mindestens 24 Satelliten, die in sechs Bahnebenen mit jeweils vier Satelliten die Erde in einer Höhe von rund 20.200 km umkreisen. Die Satellitenbahnen sind um 55° zur Äquatorebene geneigt. Die Satellitenbahnen sind nahezu kreisförmig, die Umlaufzeit eines Satelliten beträgt 11 Stunden 58 Minuten. Mit 24 Satelliten können von jedem Punkt der Erde zu jeder Zeit zwischen vier und acht Satelliten mit einem Höhenwinkel von zumindest 15° beobachtet werden.

Die wesentliche Aufgabe der Satelliten ist das Senden von Signalen, die mit geeigneten Empfängern registriert werden können. Hierzu ist jeder Satellit mit einer Uhr (Oszillator), Mikroprozessor, Sender und Antenne ausgestattet. Zustätzlich befinden sich mehrere Reserveuhren an Bord der Satelliten.

Der Oszillator im Satelliten generiert die fundamentale Frequenz von 10.23 MHZ. Ganzzahlige Multiplikation der fundamentalen Frequenz mit 154 bw. 120 erzeugt zwei Trägerwellen im L-Band, die mit L1 und L2 bezeichnet werden und die Frequenzen L1 = 1575,42 MHz und L2 = 1227,60 MHz aufweisen. Die Verwendung von zwei Trägerwellen ermöglicht die Elimination bestimmter Fehlereinflüsse.

Auf diese Trägerwellen sind zwei Codes, der C/A-Code (Coarse/Acquisition-Code) und der P-Code (Precision-Code) aufmoduliert. Die Codes stellen Zeitmarken dar und erlauben die Bestimmmung des Zeitpunktes der Signalaussendung. Der C/A-Code hat eine Wellenlänge von ungefähr 300 m und ist nur auf die L1-Trägerwelle aufmoduliert. Der P-Code hingegen hat eine Wellenlänge von etwa 30 m und ist sowohl auf die L1-Trägerwelle als auch auf die L2-Trägerwelle aufmoduliert. Durch AS (Anti-Spoofing) wird der P-Code mit dem nur autorisierten Anwendern zugänglichen W-Code überlagert; das Ergebnis der Überlagerung wird als Y-Code bezeichnet.

Der C/A-Code, auch als *Pseudo Random Noise* (PRN-Code) bezeichnet, ist für jeden Satelliten eindeutig und wird von GPS Geräten zur Identifizierung der Satelliten verwendet. Der C/A-Code wiederholt sich alle 1023 bits (eine Millisekunde). Er ist die Basis für alle zivilen GPS-Anwendungen. Die Positionsbestimmung unter Verwendung des C/A-Codes wird auch als *Standard Positioning Service* (SPS) im Gegensatz zum *Precise Positioning Service* (PPS) bezeichnet, bei dem auch der Y-Code bekannt sein muß.

Schließlich wird auf die beiden Trägerwellen L1 und L2 die sogenannte Navigationsnachricht aufmoduliert. Sie enthält die Bahndaten der Satelliten, Zeitkorrektur-Codes und andere Systemparameter.

Das Kontrollsegment

Das Kontrollsegment besteht aus einer Hauptkontrollstation (*master control station*), fünf Monitorstationen und drei Bodenkontrollstationen. Diese Stationen sind weltweit auf fünf Standorte verteilt: Colorado Springs (Zentrale und Monitorstation), Ascension, Diego Garcia und Kwajalein (jeweils Monitor- und Kontrollstationen) und Hawaii (Monitorstation).

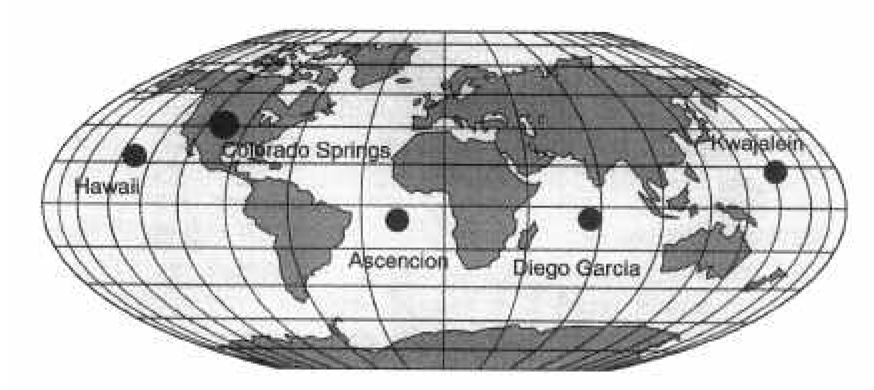


Abb. 5. Die Standorte der Bodenkontrollstationen des GPS

Zu den Aufgaben des Kontrollsegmentes gehören die Vorausberechnung der Satellitenbahnen, die Überwachung der Satellitenuhren (Gang, Stand), die Übermittlung der Navigationsnachricht an die Satelliten, sowie die Gesamtkontrolle des Systems. Dazu sind

neben Bahnkorrekturen auch SA (Selective Availability) und A-S (Anti-Spoofing) zu zählen.

SA (Selective Availability)

Unter SA versteht man die vom amerikanischen Verteidigunsministerium durchgeführte Reduzierung der erreichbaren Genaugikeit in der Echtzeit-Navigation. Diese Einschränkung der erreichbaren Genauigkeit wird einerseits durch eine Manipulation der Satellitenuhr und andererseits durch eine geringere Genauigkeit der Ephimeriden (Satellitenbahndaten) erzielt.

Das Maß der Genauigkeit kann vom Kontrollsegment gesteuert werden. Ohne SA wird mit dem C/A-Code eine Genauigkeit von etwa 15-30 m für die Positionsbestimmung erreicht. Mit dem derzeit wirksamen SA wird mit einer Zuverlässigkeit von 95% eine Lagegenauigkeit von 100 m und eine Höhengenauigkeit von 156 m garantiert.

A-S (Anti-Spoofing)

Unter A-S versteht man die Verschlüsselung des P-Codes. Der daraus resultierende „Y-Code" steht nur mehr autorisierten Anwendern zur Verfügung. Wörtlich bedeutet „Spoofing" ein Beschwindeln, somit ist A-S eine Maßnahme gegen ein „Beschwindeln".

Das Benutzersegment

Das Benutzersegment umfaßt die Menge alle GPS-Empfänger, die uneingeschränkt und kostenlos dieses System zeit-, sicht- und wetterunabhängig nutzen können. Aufgrund der Tatsache, daß es sich um ein passives System handelt, ist die Anzahl der Benutzer uneingeschränkt.

Nachstehende Grafik zeigt das Blockschaltbild eines GPS-Empfängers:

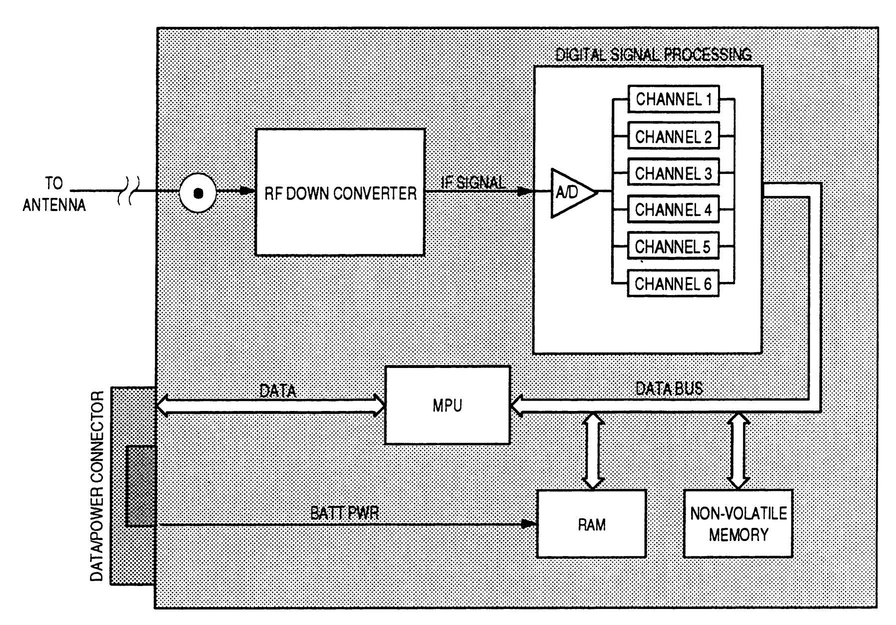


Abb. 6. Blockschaltbild eines 6-Kanal GPS-Empfängers

Die am Empfängereingang anstehenden Signale werden zunächst in einem analogen Empfangsteil des GPS-Empfängers verstärkt und auf eine niedrigere Frequenz umgesetzt.

Bis hierher ähnelt noch alles einem ganz normalen Kommunikationsempfänger und das Signal enthält noch unverändert alle Daten. Im anschließenden Signalverarbeitungsteil erfolg die Signaltrennung (in einzelne Kanäle) und die weitere Aufbereitung ist nur noch digital.



Abb. 7. Der GeoTracker von Communication & Navigation (Foto: C&N) als Beispiel für einen GPS-Empfänger

Das Prinzip der Positionsbestimmung

Die geodätisch interessanteste Komponente, die dreidimensionale Positionsbestimmung, beruht bei GPS im Prinzip auf Streckenmessungen. Sollen im dreidimensionalen Raum die Koordinaten eines unbekannten Punktes durch Streckenmessungen bestimmt werden, so sind die Strecken zwischen drei bekannten Punkten und dem unbekannten Punkt zu messen. Geometrisch liegt der Schnitt dreier Kugelschalen vor, da im dreidimensionalen Raum der geometrische Ort einer Streckenmessung mit einem bekannten Punkt eine Kugelschale ist. Zwei Kugelschalen schneiden sich in einem Kreis und dieser durchstößt die dritte Kugelschale in zwei Punkten. Davon ist einer der koordinatenmäßig gesuchte Punkt.

Bei GPS werden die bekannten Punkte durch Satelliten realisiert. Die Satellitenbahndaten (Ephimeriden), aus denen man für einen beliebigen Zeitpunkt die Koordinaten des Satelliten berechnen kann, werden bekanntlich auf das vom Satelliten gesendete Signal aufmoduliert und stehen daher dem Anwender jederzeit zur Verfügung. Die Distanz zwischen dem Satelliten und dem unbekannten Punkt kann durch Messung der Laufzeit des Radiosignals abgeleitet werden.

Sorgt man dafür, daß der im Empfänger generierte C/A-Code exakt zur selben Zeit erzeugt wird, wie der des gesuchten Satelliten, und verzögert man diesen so lange, bis die Autokorrelation ihr Maximum erreicht, so entspricht diese Verzögerungszeit genau der Laufzeit des Satellitensignales. Zwar kann man die hochgenauen Satellitenuhren als synchron laufend annehmen, man muß jedoch den Empfängeruhrfehler als Unbekannte berücksichtigen. Damit liegen aber nunmehr für eine räumliche Positionsbestimmung nicht

mehr nur die drei unbekannten Koordinaten des zu bestimmenden Punktes vor, sondern es kommt als vierte Unbekannte noch der Uhrfehler des Empfängers hinzu, der auch als unbekannte Additionskonstante in allen gleichzeitig gemessenen Entfernungen interpretiert werden kann. Daher werden die Meßgrößen als Pseudoentfernungen bezeichnet.

Um nun die Position des Empfängers zu erhalten verwendet man einen einfachen trigonometrischen Trick. Am besten läßt sich die Lösung des Uhrenproblems an einem zweidimensionalen Beispiel zeigen. Auf die dreidimensionale Positionsbestimmung ist es analog anwendbar.

Wenn wir annehmen, daß unsere Empfängeruhr eine Sekunde vor geht, erhalten wir folgende Situation mit drei Satelliten: Die drei gemessenen Standlinien ergeben die drei Schnittpunkte B, C und D.

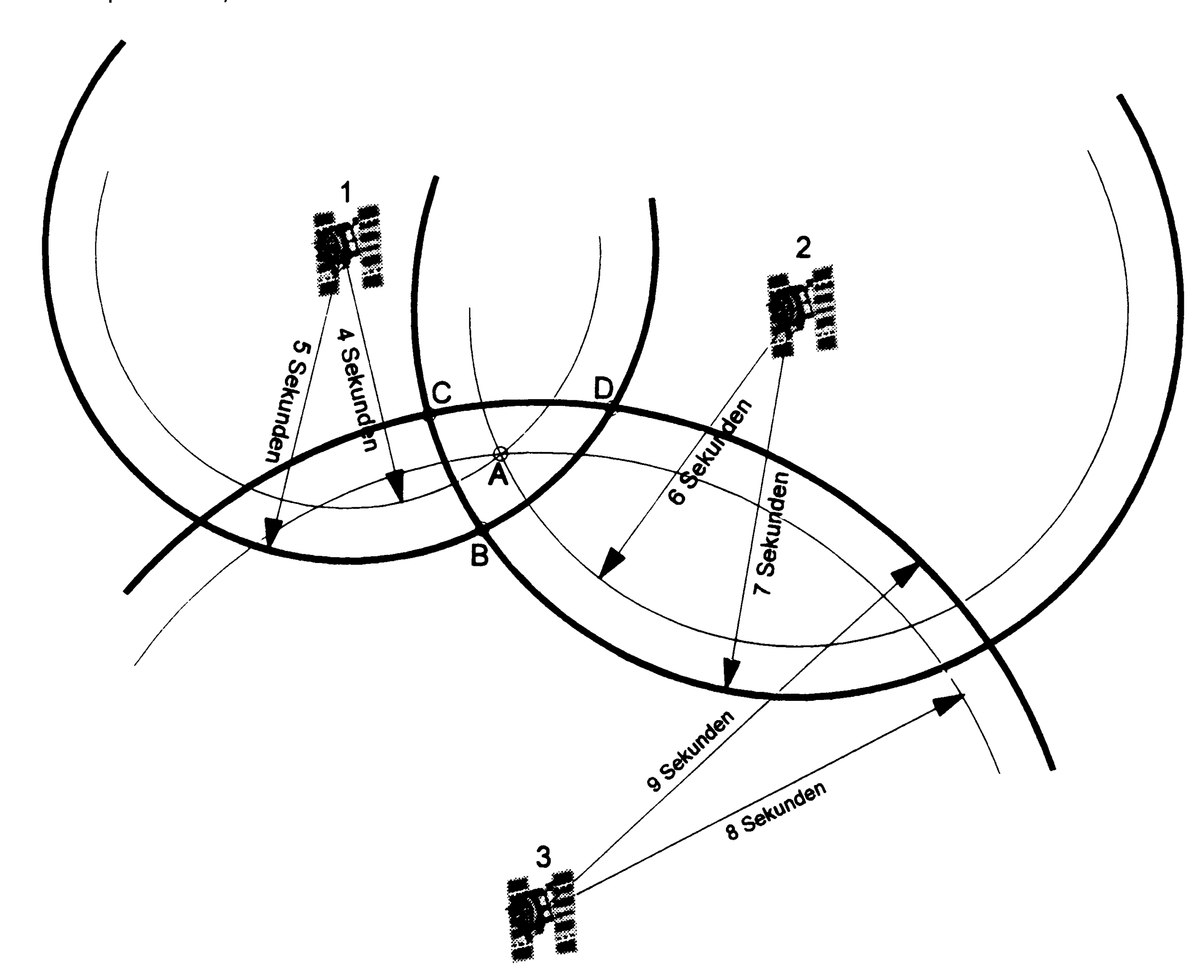


Abb. 8. Das Prinzip der Positionsbestimmung

Nun kann man bereits graphisch erkennen, wie sich die tatsächliche Position A trotz falschgehender Uhr nicht verändert, wenn man sie in den Schwerpunkt des Fehlerdreiecks BCD setzt. Dafür müssen nur zwei Voraussetzungen erfüllt sein: die Uhr muß für die Dauer der drei Messungen eine kurzzeitige, jedoch hochpräzise Genauigkeit aufweisen und es muß immer ein zusätzlicher Satellit zur Positionsbestimmung mit herangezogen werden, also vier Satelliten für eine dreidimensionale Positionsbestimmung.

Um die vier Unbekannten zu bestimmen, braucht man somit vier Pseudoentfernungen. Wegen der Forderung nach der Verfügbarkeit von GPS zu jeder Zeit und an jedem Ort

müssen daher zumindest vier Satelliten zu jeder Zeit (24 Stunden täglich) an jedem Ort gleichzeitig beobachtbar sein.

Aus den erhaltenen Psuedoentfernungen und den in der Navigationsnachricht enthaltenen Ephimeriden (Bahndaten der Satelliten) berechnet der GPS-Empfänger letztendlich die tatsächliche Position.

Genauigkeit des GPS-Systems

Zivilen Anwendern ist steht das *Standard Positioning Services* (also die Positionsbestimmung unter Verwendung des C/A Codes) weltweit frei zur Verfügung. Die Genauigkeit wird allerdings, wie bereits erwähnt, vom DOD absichtlich verringert (Selective Availability).

Standard vs. Precise Positioning Service

Das DOD garantiert mit einer Zuverlässigkeit von 95 % für das SPS eine Genauigkeit wie folgt:

- Lagegenauigkeit von 100 m, eine
- Höhengenauigkeit von 156 m und eine
- Zeitgenauigkeit von 340 Nanosekunden.

Die Genauigkeiten des Precise Positioning Service, das in erster Linie militärischen Zwecken vorbehalten ist, wird mit folgenden Werten angegeben:

- 22 m horizontale Genauigkeit,
- 27,7 m vertikal Genauigkeit und
- 100 Nanosekunden.

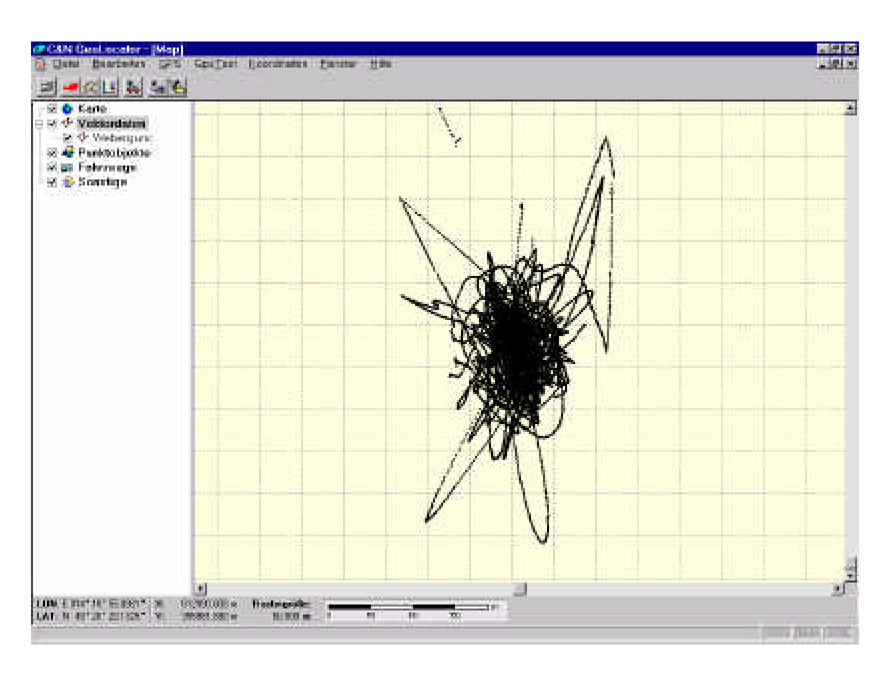


Abb. 9. zeigt eine vom Autor durchgeführte Messung über den Zeitraum von ca. 24 Stunden Man erkennt deutlich, daß durchaus auch größere, als die eingangs angegebenen Abweichungen vorkommen können (Rastergröße 50 m)..

Fehlereinflüsse

GPS-Messungen werden neben der bereits erwähnten Selective Availability noch durch verschiedene andere äußere Einflüsse verfälscht.

Geometric Dilution of Precision (GDOP)

Die Genauigkeit einer Messung hängt wesentlich von der Satellitenkonstellation, also der Entfernung und Richtung der einzelnen Satelliten zum Beobachtungsort, ab. Je ungleichmäßiger die Verteilung der Satelliten ist, desto schlechter ist die erreichbare Positionsgenauigkeit. Diese Abweichung der Genauigkeit wird als GDOP (Geometric Dilution Of Precision) bezeichnet.

Einfluß der Ionosphäre

Die vom Satelliten ausgesandten Signale müssen Schichten ionisierter Gasatome durchqueren, welche die Ausbreitungsgeschwindigkeit dieser Signale hemmt. Die Ionenkonzentration ist kein konstant kalkulierbarer Faktor, sondern hängt stark von der Tageszeit, Jahreszeit, Sonnenaktivität, geographischer Breite des Empfängers und von der Höhe und Richtung des Satelliten ab. Bei Empfängern, welche beide Trägerwellen L1 und L2 empfangen können, kann der Einfluss der Ionosphäre kompensiert werden.

Einfluß der Troposphäre

Auch die Troposphäre beeinflusst die Ausbreitung der Satellitensignale (troposphärische Refraktion). Entscheidend ist hierbei der Höhenwinkel der Satelliten.

Multipath

Dieser Effekt tritt infolge von Mehrfachreflexionen des Signals auf. Diese Mehrwegausbreitung des Satellitensignals tritt vor allem in der Nähe von reflektierenden Flächen auf und kann die gemessenen Phasen bis zu mehreren Zentimetern verfälschen. Multipath kann durch spezielle Antennen oder eine geeignete Wahl des Antennenstandortes reduziert oder ganz vermieden werden. Auch Sender oder sonstige elektrische Anlagen (Hochspannungsleitungen, Oberleitungen, Transformatoren) in unmittelbarer Nähe der Antenne können die Satellitensignale beeinflussen. Multipath-Effekte treten auch in stark bewaldetem Gebiet auf.

Genauigkeit der Satellitenumlaufbahnen

Zwischen der vorausberechneten und der tatsächlichen Umlaufbahn der Satelliten gibt es geringfügige Abweichungen, den sogenannten Ephemeridenfehler.

Genauigkeit der Uhren

Auch die hochpräzisen Atomuhren sind nicht frei von Abweichungen.

Abschattung der Signale durch Horizontüberhöhung

Abschattung tritt unter dicht belaubten Bäumen, unter einer überhängenden Felswand und in engen Straßenschluchten auf. Dadurch ist der gleichzeitige Empfang von vier verschiedenen Satellitensignalen erschwert und somit keine Positionsbestimmung möglich.

Verfahren zur Verbesserung der Genauigkeit

Phasenmessung

Für Anwendungen (z.B. Geodäsie, Vermessungswesen), für die die Genauigkeit des SPS nicht ausreicht, benötigt man Empfänger, die neben den Signallaufzeiten auch Messungen der Phasen der Trägerwellen erlauben. Dabei unterscheidet man zwischen Einfrequenz- und Zweifrequenzempfänger, je nachdem ob die Phasen einer oder beider Trägerwellen registriert werden können. Dies kann bei Kenntnis des C/A- oder Y-Codes über die Code-Korrelation erfolgen. Über den C/A-Code kann damit allerdings nur die Trägerwelle L1 wiederhergestellt werden. Zur Rekonstruktion beider Trägerwellen L1 und L2 über eine Code-Korrelation wird der Y-Code benötigt, der jedoch vom DoD geheimgehalten wird.

Bestimmte Techniken, etwa das sog. Quadrierverfahren (Squaring), erlauben zwar auch bei Nichtverfügbarkeit des Y-Codes die Nutzung der L2-Trägerwelle, allerdings kommt es dadurch zu einem Genauigkeits- oder einem völligen Datenverlust durch ein höheres Rauschen im Signal.

Eine Messung, die auf der Trägerphase (Carrier Phase) basiert, hat als Unbekannte die Phasenmehrdeutigkeit (ambiguity). Die Phasenmehrdeutigkeit ergibt sich aus der gesamten Anzahl der Wellenzyklen in der Entfernung Satellit - Beobachter. Wenn es gelingt diese Fehlereinflüsse zu beherrschen, so läßt sich auch das hohe Genauigkeitspotential von GPS ausschöpfen und es somit für Ingenieurvermessungen nutzen. Mit Hilfe der Phasenmessung sind Meßgenauigkeiten vom cm- bis mm-Bereich möglich.

Differential GPS (DGPS)

Die Positionsgenauigkeit, die man mit einem Differential-GPS-System erreichen kann, liegt unter einem Meter. Wertet man die Phase der Trägerfrequenz mit aus, sind sogar Positionsbestimmungen im Zentimeterbereich möglich.

Aufgrund der Tatsache, daß der Fehler zwischen der gemessenen und der wirklichen Position in einem Gebiet von mehreren zehntausend Quadratkilometern annähernd gleich groß ist, läßt sich DGPS folgendermaßen realisieren: Man positioniert einen GPS-Empfänger ortsfest an einem exakt vermessenen Punkt. Die Positionsdifferenz zwischen diesem bekannten Ort und dem gerade ermittelten Meßwert wird in Echtzeit, d.h. sofort, über eine Datenfunkstrecke an den Navigierenden übermittelt. Dessen GPS-Empfänger korrigiert mit diesen Daten seine eigenen, ebenfalls in Echtzeit gemessenen Positionsdaten. Voraussetzung ist, daß beide Empfänger mindestens vier identische Satelliten empfangen.

Ein bereits realisiertes Protokoll mit der Bezeichnung RTCM 104 (Real-Time-Correction-Message) wird von allen modernen, DGPS-fähigen Empfängern verstanden und eine Korrektur kann somit automatisch durchgeführt werden. Voraussetzung dafür ist allerdings ein Anbieter, der die RTCM-Daten per Datenfunk zur Verfügung stellt. In Österreich bietet die DGPS-Datenverarbeitungs GmbH in Zusammenarbeit mit ORF seit 15. Mai 1998 gegen Lizenzgebühr ein DGPS-Service an.

Postprocessing

Bei dieser Art der Positionsbestimmung wird die exakte Position erst im nachhinein bestimmt. Für Echtzeitanwendungen scheidet diese Möglichkeit zwar aus, im Bereich Vermessungswesen, ist dieses Verfahren jedoch weit verbreitet. Im Unterschied zu DGPS werden die Korrekturdaten vom ortsfesten Empfänger nicht an den Feldempfänger übertragen, sondern die gemessenen Positionen nach erfolgter Erfassung in einen PC übertragen und die exakten Positionswerte mittels geeigneter Postprocessing-Software berechnet.

Inverses Postprocessing

Dabei handelt es sich um eine neue Technik, die derzeit noch kaum eingesetzt wird, obwohl das Prinzip dem des Differential GPS sehr ähnlich ist. Der wesentliche Unterschied besteht darin, daß der Feldempfänger (Rover) nicht seine Positionen zum Leitstand schickt, sondern seine Satelliten-Rohdaten. Die Position wird erst im Leitstand mit Hilfe der Positionsdaten des ortsfesten Empfängers berechnet. Vorteil dabei ist, daß man als Feldempfänger auch billigere Geräte einsetzen kann, ohne dabei einen merklichen Qualitätsverlust hinnehmen zu müssen. Nachteil dabei ist allerdings, daß die Position nur im Leitstand und nicht vor Ort verfügbar ist.

Zivile Anwendungsgebiete

Neben der militärischen Nutzung hat sich die GPS-Technologie bereits in vielen zivilen Bereichen etabliert. So wird sie inzwischen auf folgenden Gebieten eingesetzt:

- Navigation für Luft- und Seefahrt
- Vermessungstechnik
- GIS-Datenerfassung
- Fahrzeugortung, Logistik, Ver- und Entsorgungsbetriebe
- Leitsysteme für Rettung, Polizei, Feuerwehr und Katstrophenschutz
- technische Geologie, Erderkundung (Plattentektonik, ...)
- Bestandsaufnahmen
- Bergwesen
- Archäologie
- Topographie
- Hydrographie
- Kulturtechnik und Wasserwirtschaft, Planung von Versorgungsleitungen
- Forst- u. Holzwirtschaft, landwirtschaftliche Flächenbestimmung
- Umwelttechnik
- Zeitmeßtechnik
- Blindenhilfe

Kapitel 2:
GSM – Global System For Mobile Communications

Einleitung

Damit ein Fahrzeug *geortet* werden kann, d.h. daß die Fahrzeugposition an einem beliebigen anderen Punkt bekannt ist, müssen die GPS-Daten an den zentralen Leitstand übertragen werden. Dazu bieten sich grundsätzlich folgende Möglichkeiten an:

- Datenübertragung per Funk
- Datenübertragung per Satellitenverbindung:
  International Mobile Satellite Organization (INMARSAT)
- Datenübertragung per GSM

Funk hat nur eine beschränkte Reichweite. In Österreich sind Funkmodems mit einer maximalen Sendeleistung von 0,5 Watt zulässig. In der Praxis erreicht man damit eine Reichweite von höchstens 5 bis 10 km Entfernung. Sind größere Strecken zu überbrücken, scheidet diese Möglichkeit also aus.

Für weltweite Datenübertragung bietet *INMARSAT* die Möglichkeit der Übertragung per Satelliten an. Diese Variante ist allerdings sowohl von der technischen Ausrüstung, als auch von den laufenden Kosten her, im Vergleich zu GSM relativ teuer.

Besondere Bedeutung kommt nun GSM, dem Global System For Mobile Communications, zu. Die GSM-Technologie ist inzwischen weit verbreitet, zur Datenübertragung geeignete Geräte sind für jedermann erschwinglich.

Neben der Möglichkeit mittels eines *Data Call* eine stehende Punkt-zu-Punkt-Verbindung zwischen GSM-Geräten aufzubauen, kommt dem *Short Messaging Service* (SMS), ein spezieller Dienst unter GSM, für die GPS-Datenübertragung besondere Bedeutung zu. In diesem Fall besteht keine ständige Verbindung zum Fahrzeug, was im Regelfall auch nicht benötigt wird. Das zu ortende Fahrzeug soll nur in bestimmten Intervallen seine Position an den Leitstand übermitteln.

Eine *Short Message* ist zwar auf 140 Bytes Daten beschränkt (bzw. 160 Zeichen bei Verwendung eines 7 Bit Codes) doch ist dieses Datenvolumen für die Übertragung eines GPS-Datensatzes ausreichend.

Für bestimmte Anwendungsbereiche, insbesondere im Speditionsbereich, wo nur fallweise eine Position eines Fahrzeuges benötigt wird, erweist sich das Short Messaging Service als ideale Übertragungsmöglichkeit.

Vorteile von SMS gegenüber Data Call:
- Gesicherte Übertragung.
- mehrere Fahrzeuge können quasi parallel geortet werden.

- Keine Kosten für die stehende Verbindung
- überdies bieten die viele Netzbetreiber für Geschäftskunden günstigere SMS-Tarife an.

Nachteile:

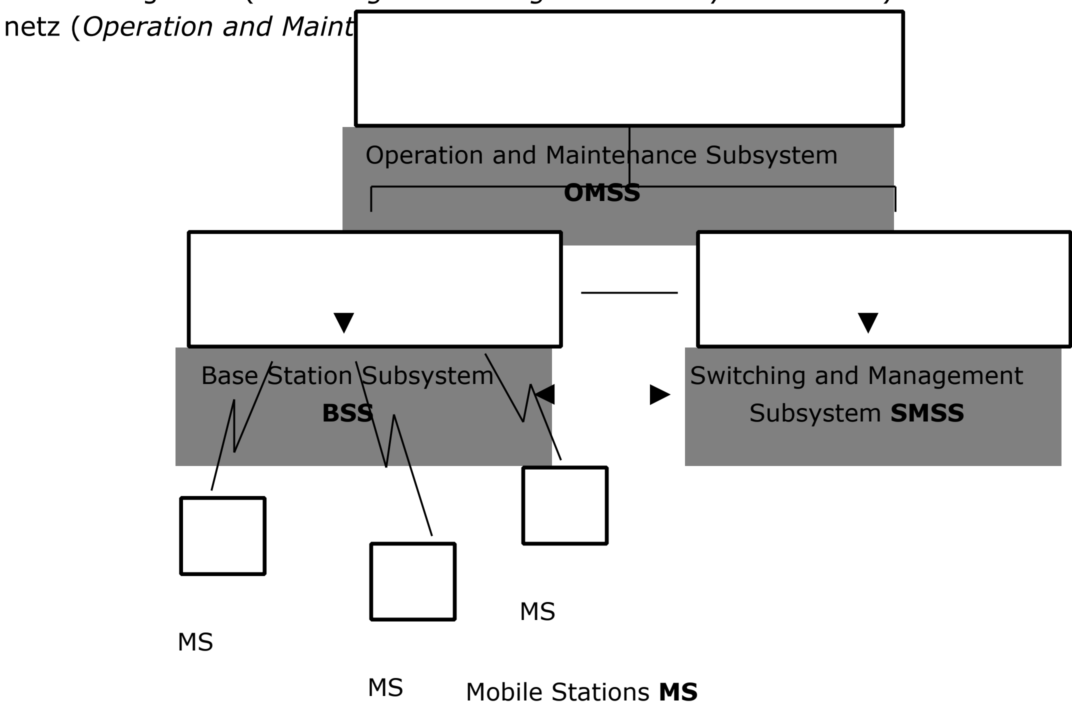
- mögliche Verzögerungen bei der Zustellung:
  Ist das GSM-Gerät gerade nicht erreichbar (ausgeschaltet, busy oder nicht im Netzbereich), kann eine Nachricht nicht zugestellt werden. In diesem Fall versucht das Messaging Center des Netzbetreibers die Nachricht erst nach einer relativ langen Zeitspanne erneut zuzustellen.

  Da diese Zeitspannen abhängig vom Netzbetreiber sind, kann an dieser Stelle keine allgemeingültige Aussage gemacht werden. (Beispiel max.mobil: zweiter Zustellungsversuch nach einer Minute, dritter Zustellungsversuch etwa erst nach einer halben Stunde).

GSM-Systemarchitektur

Ein GSM-System hat zwei wesentliche Bestandteile: die fest installierte Infrastruktur (das Netz im eigentlichen Sinne) und die Mobilteilnehmer, welche über Funk (Luftschnittstelle, *air interface*) die Dienste des Netzes nutzen und kommunizieren.

Das fest installierte GSM-Netz wiederum kann in drei Subnetze (i.A. als *Subsysteme* bezeichnet) untergliedert werden: das Funknetz (*Base Station Subsystem* BSS), das Mobilvermittlungsnetz (*Switching and Management Subsystem* SMSS) und das Managementnetz (*Operation and Maint*



Mobilstation

Mobilstationen (*Mobile Stations* MS) sind die Geräte, die von Mobilteilnehmern für den Dienstzugang genutzt werden. Sie bestehen aus zwei wesentlichen Komponenten: dem Gerät selbst (*Mobile Equipment* ME) und dem *Subscriber Identity Module* (SIM). Erst das SIM eines Teilnehmers – ausgeführt als fest eingebauter Chip oder als austauschbare Chip-Karte – macht aus einem Mobilgerät eine vollwertige Mobilstation, mit der ein Dienstzugang möglich ist. Zusätzlich zur Gerätekennung (*International Mobile Equipment Identity* IMEI) besitzt ein Mobilgerät abhängig vom eingesetzten SIM eine Teilnehmerkennung (*International Mobile Subscriber Identity* IMSI) und eine Rufnummer (*Mobile Station Integrated Systems Digital Network Number* MSISDN). GSM Mobilgeräte werden also mit dem SIM personalisiert.

Durch dieses bei GSM zum ersten Mal konsequent angewandte Konzept des SIM wird zum einen eine Trennung der Benutzermobilität von der Gerätemobilität erreicht. Internationales Roaming unbahängig von Mobilgerät und Netztechnologie wird dadurch möglich, vorausgesetzt die Schnittstelle zwischen Gerät und SIM ist normiert. Zum anderen übernimmt das SIM wesentlich mehr Aufgaben als nur die Personalisierung von Mobilgeräten mit der IMSI und MSISDN. Im SIM sind auch alle geheimzuhaltenden kryptographischen Algorithmen realisiert, die wichtige Funktionen zur Authentifizierung und Nutzdatenverschlüsselung implementieren.

Darüberhinaus können auf dem SIM Kurznachrichten (Short Messages) und Gebühreninformationen gespeichert werden und es besitzt eine Telefonbuchfunktion mit Kurzwahlliste zur Speicherung von Namen und Telefonnummern. Neben den teilnehmerspezifischen Daten können im SIM auch netzspezifische Daten gespeichert werden (z.B. Listen von Trägerfrequenzen, auf denen das Netz periodisch Systeminformationen ausstrahlt, oder auch die *Local Area ID*, LAI). Die Nutzung des SIM und damit der Mobilstation kann durch eine vierstellige PIN gegen unberechtigten Zugriff geschützt werden.

Funknetz (Base Station Subsystem BSS)

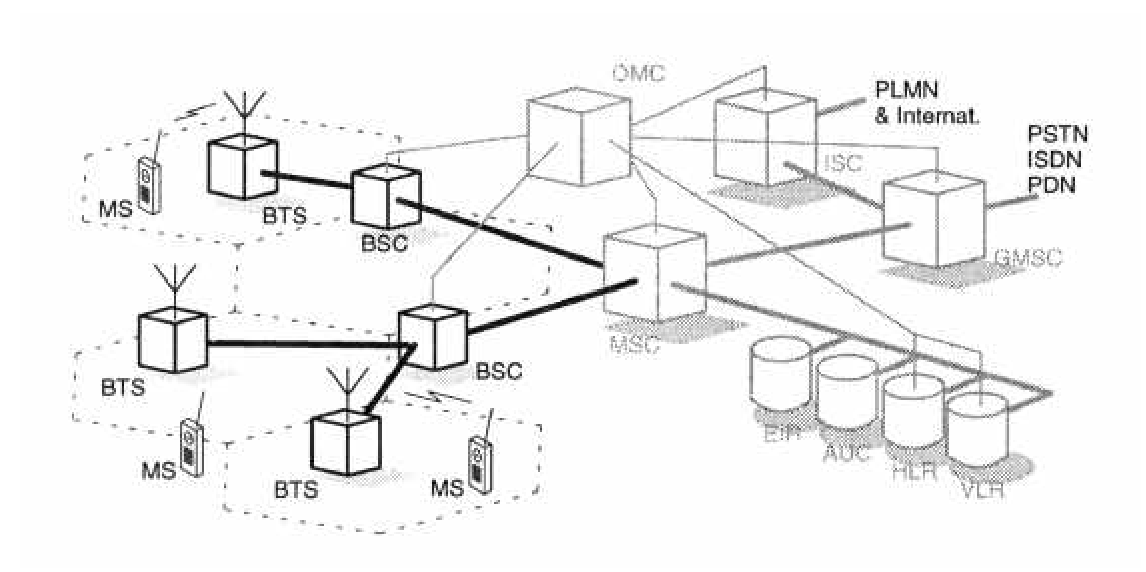


Abb. 10. Die Komponenten des Base Station Systems:
BTS Base Transceiving Station
BSC Base Station Controller
MS Mobile Station

Eine GSM-Zelle wird vom Funkbereich einer BTS (*Base Transceiving Station*) aufgespannt. Die BTS stellt in dieser Zelle die Funkkanäle für Signalisierung und Nutzung zur Verfügung. Sie besitzt neben dem HF-Teil nur einige wenige Komponenten zur Signal- und Protokollverarbeitung. Um die Basisstationseinheiten klein zu halten, ist die wesentliche Steuerungs- und Protokollintelligenz in den Basisstations-Controller BSC (*Base Station Controller*) verlagert. Im BSC wird beispielsweise das Handover-Protokoll abgewickelt. Mehrere BTS können gemeinsam von einem BSC gesteuert werden.

An der Funkschnittstelle werden zwei Typen von Kanälen bereitgestellt: Verkehrs- und Signalisierungskanäle. Für die Verkehrskanäle umfaßt das BSS im wesentlichen alle Funktionen der OSI Schicht 1.

Mobilvermittlungsnetz (Switching and Management Subsystem SMSS)

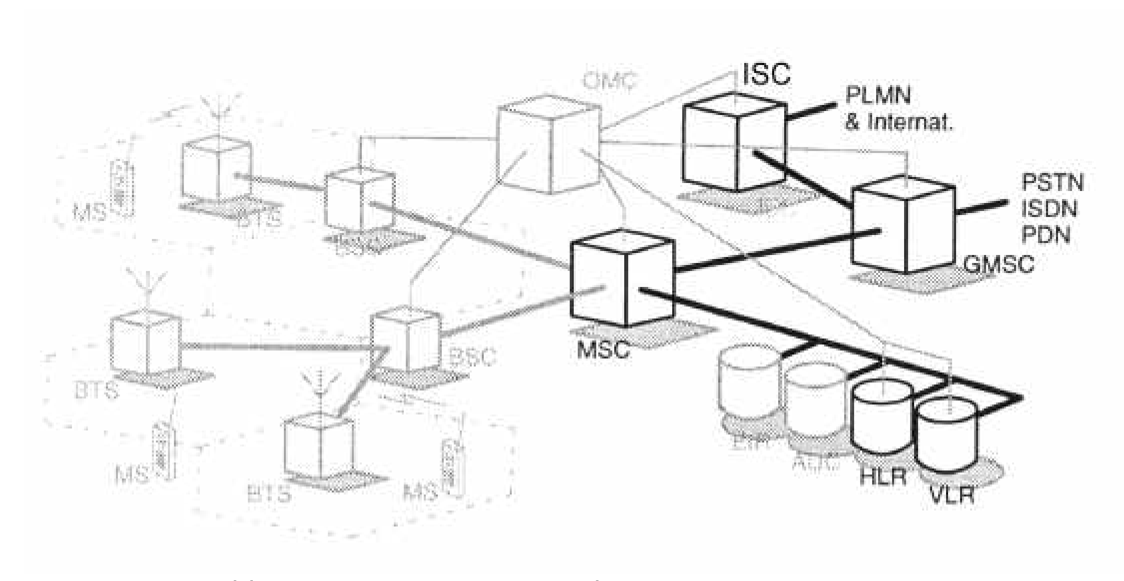


Abb. 11. Die Komponenten des SMSS:
MSC Mobile Switching Center
HLR Home Location Register
VLR Visited Location Register
GMSC Gateway MSC
ISC International Switching Center

Das Mobilvermittlungsnetz (*Switching and Management Subsystem* SMSS) besteht aus einem oder mehreren Mobilvermittlungszentren (Mobile Switching Center MSC) und den Datenbanken, welche die zur Vermittlung und Diensterbringung notwendigen Daten speichern: Heimatregister (*Home Location Register* HLR) und Besucherregister (Visited *Location Register* VLR)

Mobilvermittlungszentrum (Mobile Switching Center)

Das MSC erfüllt alle vermittlungstechnischen Funktionen eines Festnetz-Vermittlungsknotens, wie Wegsuche, Signalwegschaltung und Dienstmerkmalsbearbeitung. Der Hauptunterschied zwischen einem ISDN-Vermittlungsknoten und einem MSC ist, daß ein MSC die Zuteilung und Verwaltung von Funkressourcen und die Mobilität der Teilnehmer zu berücksichtigen hat. Das MSC hat deshalb zusätzliche Funktionen für die Aufenthaltsregistrierung von Teilnehmern und für den Handover von Verbindungen beim Wechsel von Zellen vorzusehen.

Jedes GSM PLMN (*Public Land Mobile Network*) kann mehrere MSC besitzen, von denen jedes für einen Teil des Dienstgebietes (*service area*) zuständig ist.

Zum Verbindungsaufbau des Gesprächsverkehrs in und aus dem Festnetz stehen eigene Gateways (Gateway MSC, GMSC) zur Verfügung. Verbindungen zu anderen Mobilnetzen und andere internationale Verbindungen werden meist über das ISC (International Switching Center) des jeweiligen Landes geroutet.

Heimat- und Besucherregister (HLR und VLR)

Zur Teilnehmerregistrierung und –lokalisierung sind zwei funktionale Einheiten innerhalb eines GSM PLMNs definiert: das Heimatregister HLR (Home Location Register) und das Besucherregister VLR (Visited Location Register). Das HLR ist das Register, das jeden Teilnehmer und jede Mobil-ISDN-Rufnummer registriert, die in seinem Netz „beheimatet" sind. Es speichert alle relevanten Teilnehmerdaten. Zu den gespeicherten Informationen gehört neben festen Einträgen wie abonnierten Diensten und Berechtigungen vor allem auch ein Verweis auf den aktuellen Aufenthaltsort einer Mobilstation.

Das VLR speichert die Daten aller Mobilstationen, die sich momentan im Verwaltungsbereich des zugehörigen MSC aufenthalten. Ein VLR kann für das Gebiet eines oder mehrerer MSC verantwortlich sein. Mobilstationen können sich frei bewegen (roaming) und damit je nach Aufenthaltsort in den VLR ihres Heimatnetzes eingebucht sein, oder auch in den VLR „fremder" Netze anderer Betreiber, sofern zwischen den Betreibern ein Roaming-Abkommen besteht. Eine Mobilstation startet dazu jeweils beim Betreten einer *Location Area* LA eine Registrierungsprozedur. Das zuständige Vermittlungszentrum (MSC) leitet dabei die Identität der Mobilstation und ihre momentane *Location Area Identity* (LAI) an das Besucherregister weiter, das diese Werte in seine Datenbasis einträgt und damit die Mobilstation registriert. Wenn die Mobilstation noch nicht in diesem VLR registriert war, wird das HLR über den aktuellen Aufenthaltsort der MS informiert. Dabei werden Informationen an das HLR übergeben, welche die Wegsuche für Rufe zu dieser Mobilstation ermöglichen.

Betrieb und Wartung
(Operation and Maintenance Subsystem OMSS)

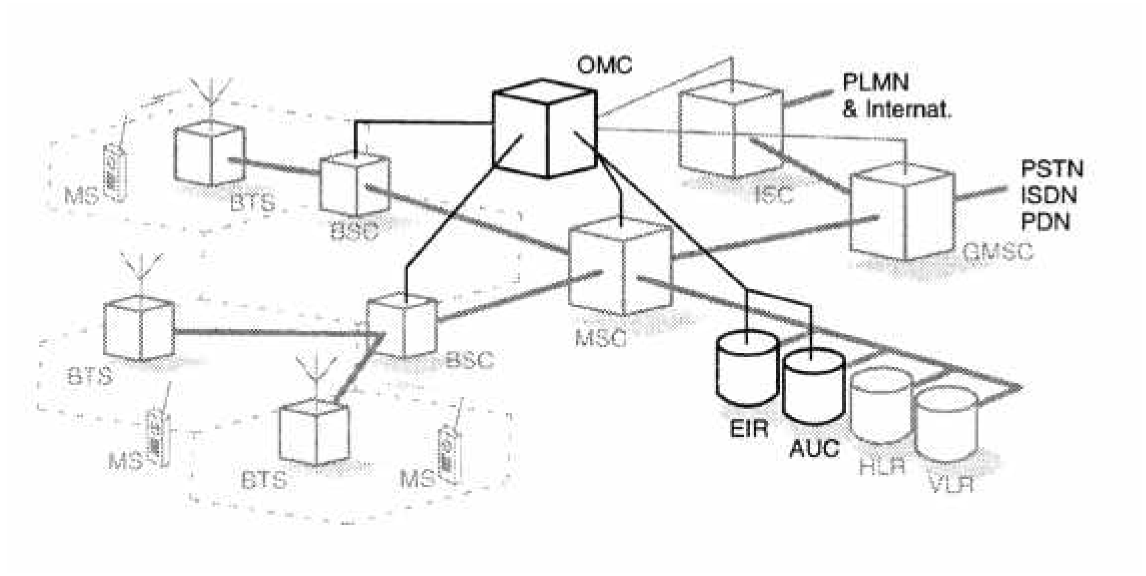


Abb. 12. Die Komponenten des OMSS:
OMC Operation and Maintenance Center
EIR Equipment Identity Register
AUC Authentication Center

Der laufende Netzbetrieb wird mit dem *Operation and Maintenance Subsystem* OMSS gesteuert und gewartet. Zu den Funktionen des OMC (*Operation and Maintenance Center*) gehören:

- Verwaltung und kommerzieller Betrieb
  (Teilnehmer- und Endgeräteverwaltung, Abrechnungen, Statistik, ...)
- Sicherheitsmanagement
- Netzkonfiguration, Netzbetrieb und Performance Management
- Wartungsarbeiten

Neben HLR und VLR sind im GSM noch zwei weitere Datenbanken vorhanden: das AUC (*Authentication Center*) und das Geräteregister EIR (*Equipment Identity Register*). Diese beiden Datenbanken dienen der Teilnehmeridentifikation und –authentifizierung sowie zur Geräteregistrierung. Im AUC werden vertrauliche Daten und Schlüssel gespeichert bzw. erzeugt. DAS EIC speichert die herstellerabhängigen Seriennummern der Endgeräte (IMEI), so daß z.B. jede Mobilstation auf veraltete Software geprüft oder einem bestimmten, als gestohlen gemeldeten Endgerät der Dienstzugang gesperrt werden kann.

Dienste des GSM

Die GSM-Dienste sind in drei Kategorien aufteteilt, und zwar in Trägerdienste (*Bearer Services*), Telematikdienste (*Teleservices*) und Zusatzdienste (*Supplementary Services*).

Die folgenden drei Kapitel geben einen kurzen Überblick über diese Dienste. Aufgrund seiner besonderen Bedeutung im Zusammenhang mit GPS-Ortung habe ich dem Kurznachrichtendienst SMS (*Short Message Service*) - einem speziellen Telematikdienst – im Anschluß daran ein eigenes Kapitel gewidmet.

Trägerdienste

Die Trägerdienste stellen die grundlegenden technischen Möglichkeiten zur Übertragung von binär vorliegenden Daten zur Verfügung. Sie bieten asynchorne und synchrone Datentransportmöglichkeiten mit leitungs- oder paketorientierter Vermittlung und Datenraten von 300 bit/s bis zu 9,6 kbit/s bzw. 13 kbit/s. Der Trägerdienst mit einer Bitrate von 13 kbit/s ist allerdings nur zur Sprachübertragung vorgesehen.

Trägerdienste stellen nur den anwendungsunabhängigen Informationstransport innerhalb des GSM-Netzes sicher. Sie entsprechen damit den Schichten 1 und 2 des OSI-Schichten-Modelles. Betreiber von Endeinrichtungen TEs (*Terminal Equipment*) können unter Nutzung dieser Trägerdienste beliebige Protokolle höherer Schichten einsetzen, sind für die Kompatibilität der in den TEs betriebenen Protokollen aber selbst verantwortlich. Bei den *Telematikdiensten*, sind auch die Protokolle in den Endeinrichtungen standardisiert worden.

Die Trägerdienste zur Datenübertragung in GSM werden in zwei verschiednen Modi angegeben: transparent (T) oder nicht-transparent (NT). Die transparente Verbindung ist durch Vorwärtsfehlerkorrektur gesichert. Abhängig vom jeweiligen Kanalzustand existiert eine schwankende Bitfehlerrate. Der nicht-transparente Modus aktiviert für die zusätzliche Sicherung der Datenübertragung das speziell auf den GSM-Funkkanal angepaßte *Radio Link Protocol* RLP. Durch das ARQ-Verfahren des RLP werden Blöcke mit Restbitfehlern, die nach der Vorwärtsfehlerkorrektur eventuell noch im Datenstrom enthalten sind, zur Wiederübertragung angefordert.

Telematikdienste

Aufbauend auf die Trägerdienste in GSM ist eine Reihe von Telematikdiensten (*teleservices TS*) definiert. Die wichtigsten Kategorien sind: Sprache, Kurznachrichtendienste (SMS), Zugang zu Message Handling Systemen (MHS-Zugang) und zum Bildschirmtext-System, sowie Teletext- und Fax-Übertragung.

Die Sprachdienste waren von jedem Netzbetreiber in der Anfangsphase bis 1991 zu implementieren. Im folgendende Tabelle enthält einen Auszug der Telematikdienste:

Kategorie	**TS Nr.**	**Dienst**
Sprache	11	Telefondienst
	12	Notruf
Kurznachrichtendienste	21	Short Message Mobile Terminated, Point to Point
	22	Short Message Mobile Originated, Point to Point
	23	Short Message Cell Broadcast
MHS-Zugang	31	Zugang zu Message Handling Systemen
Videotext Zugang	41	Videotext Zugang Profil 1
	42	Videotext Zugang Profil 2
	43	Videotext Zugang Profil 3
Teletextübertragung	51	Teletext
Faxübertragung	61	Sprache und Fax Gruppe 3 alternierend
	62	Fax Gruppe 3 automatisch

GSM Telematikdienste

Telematikdienste wie etwa der Zugang zu einem Message Handling System (X.400) oder zum Videotext-System sind nur optional vorgesehen. Mangels Nachfrage existieren auch kaum Implementierungen dieser Dienste bei den Netzbetreibern. Dafür wurde der Ausbau der asynchronen Datendienste in vielen GSM-Netzen vorangetrieben, so daß mittlerweile fast weltweit auch von mobilen Rechnern aus Faxversand/-empfang und Datenkommunikation möglich sind.

Ein weiterer Telematikdienst, dem eine hohe Priorität eingeräumt wurde, ist die Möglichkeit an der Mobilstation Kurznachrichten zu empfangen und zu versenden (Short Message Service SMS, Teleservice Nr. 21 und 22).

Das Short Message Service (SMS)

Dieser Dienst sollte seit 1996 in allem GSM-Netzen zur Verfügung stehen. Der Dienst TS21 ermöglicht es, gezielt einzelnen Mobilstationen eine bis zu 160 Zeichen[5] lange Nachricht zu senden. Umgekehrt erlaubt Dienst TS22 Mobilstationen auch Kurznachrichten zu versenden. Mit dem Kurznachrichtendienst in Verbindung mit Mehrwertdiensten – beispielsweise Mailboxsysteme mit automatischem Versand einer Kurznachricht bei gespeicherten Anrufen – geht das Dienstangebot der GSM-Netze über den Umfang der Dienste in den Festnetzen hinaus.

[5] Vorausgesetzt, es wird ein 7-Bit-Code zur Codierung der alphanumerischen Nachricht verwendet. Alternativ dazu ist auch die Verwendung des 8 Bit ASCII-Codes möglich. In diesem Fall kann die Nachricht nur 140 Zeichen umfassen, allerdings ist diese Möglichkeit in einigen, sich derzeit am Markt befindlichen Endgeräten nicht implementiert.

Für den Kurznachrichtendienst muß ein Netzbetreiber ein Dienstzentrum (*Service Center*) einrichten, das im Store-and-Forward-Betrieb Kurznachrichten entgegennimmt (von Mobilstationen, e-mail, Fax, etc.) und diese dann gegebenenefalls zeitversetzt an den Empfänger unabhängig von seinem Aufenthaltsort weiterleitet. Umgekehrt ist es grundsätzlich möglich, Kurznachrichten auch an Festnetzkunden weiterzuleiten (per Fax, e-mail, etc.)

Die Kurznachrichten werden über ein verbindungsloses, paketvermittelndes Protokoll übertragen. Der Empfang einer Nachricht muß von der Mobilstation bzw. dem Service Center bestätigt werden. Die Übertragung von Kurznachrichten ist gesichert. Treten während der Übertragung einer Nachricht Störungen auf, wird sie wiederholt. Rückmeldungen, ob und wann eine Nachricht gelesen wurde gibt es allerdings nicht.

Die Telematikdienste des SMS (TS21 und TS22) sind die einzigen Telematikdienste, welche gleichzeitig mit anderen Diensten zusammen genutzt werden können, d.h. Kurznachrichten können auch während eines Gespräches gesendet oder empfangen werden.

Standardisierung und Weiterentwicklung

GSM ist kein abgeschlossenes System, das sich nicht mehr verändert. Die GSM-Standards werden laufend weiterentwickelt. Die Phase 1 der Implementierung von GSM-Systemene beinhaltete grundlegende Dienste, wie z.B. die Sprachkommunikation und einige wenige Zusatzdienste, die zur Markteinführung 1991 verbindlich angeboten werden konnten. In der Phase 2, deren Standardisierung 1995 abgeschlossen wurde (Markteinführung 1996) kamen vorallem Zusatzdienste hinzu.

Diese weitere Entwicklung läuft allgemein unter dem Namen GSM Phase 2+. Dabei rücken auch Trägerdienste mit höheren Bitraten bis zu 64 kbit/s in das Zentrum des Interesses. Auch verbindungslose, paketvermittelte Datenkommunikation auf der Luftschnittstellle (General Packet Radio Service GPRS) wird gegenwärtig untersucht und standardisiert. Beides ist besonders auch für die Fahrzeugortung interessant, da für diesen Anwendungsfall nicht ein kompletter Verkehrskanal für die Dauer einer Verbindung belegt werden muß.

Kapitel 3:

GeoLocator – Fahrzeugortung per GPS und GSM

Zielsetzungen

Mit dem GEOLOCATOR soll ein GPS-basiertes Fahrzeugortungssystem entstehen, das es ermöglicht die von einem GPS-Empfänger erhaltenen Daten auszuwerten, zu protokollieren und geeignet darzustellen. Als „Hintergrund" sollen sowohl digitalisierte Landkarten (auch Satellitenbilder) als auch vektororientierte Daten (Leitungssysteme, Straßen, Vermessungspunkte, zusätzliche Orte und andere Objekte, vom GPS-Empfänger aufgezeichnete Routen) verwendet werden können.

Grundsätzlich sind zwei Einsatzvarianten möglich: (a) als mobiles System, z.B. ein Laptop, an den der GPS-Empfänger direkt angeschlossen wird und (b) als ortsgebundener Leitstand, der die Verbindung zum GPS-Empfänger (meist zu einem Fahrzeug) über ein Datenfunkmodem oder über GSM aufbaut.

Zusätzlich sollen weitere ortsrelevante Informationen aufgezeichnet werden können. Die zusätzlichen Informationen können entweder direkt vom Benutzer über die Tastatur eingegeben werden (z.B. Bezeichnung des aktuellen Standortes), oder von einem Meßgerät (z.B. Meßwerte von Bodenproben, o.ä.) stammen.

Auf dem verwendeten Kartenmaterial nicht existierende Orte oder Objekte, können vom Benutzer manuell eingetragen werden. Die Objekte können zu Gruppen zusammengefaßt werden und wahlweise eingeblendet werden.

Mit Hilfe der integrierten Fahrzeugverwaltung[6] besteht die Möglichkeit ausgewählte Fahrzeuge sofort auf Befehl oder in Intervallen periodisch zu orten.

Funktionsumfang

Im Zuge der Entwicklung des GeoLocators soll ein allgemein verwendbares GPSAPI entstehen, das im wesentlichen folgende Funktionen beinhaltet:

- Kontrolle von mehreren seriellen Schnittstellen
- Lesen verschiedener GPS-Daten (Binär, NMEA, ...), herstellerabhängig

[6] Es wird hier der Begriff „Fahrzeugverwaltung" verwendet, da dies dem voraussichtlich häufigsten Anwendungsfall entspricht. Genausogut können aber auch einzelne Personen, die mit einem GSM-Handy und entsprechendem GPS-Empfänger ausgerüstet sind, geortet werden.

- Koordinatentransformation (rechnet nach Bedarf die einlangenden GPS-Koordinaten auf ein gewünschtes Zielkoordinatensystem um)
- Attributvergabe (einzelnen GPS-Positionen können Eigenschaften zugewiesen werden, Text, ...)
- Fahrzeugverwaltung (Auswahl, Namensvergabe)
- Verwaltung und Einleitung von einzelnen oder zyklischen Ortungen je Fahrzeug

Bisherige geographische Informationssysteme verwenden entweder ausschließlich digitales Kartenmaterial oder sind rein vektororientiert. Der GeoLocator soll die Vorteile beider Systeme in sich vereinen: Einerseits die rasche Verfügbarkeit von digitalisiertem Kartenmaterial (jeder beliebige Stadtplan kann mithilfe eines Scanners digitalisiert werden und ist in der Folge als Basis für den GeoLocator verfügbar) und die Genauigkeit von vektororientierten Daten (exakte Darstellung bei beliebiger Vergrößerung). Außerdem sind vom GPS-Empfänger erhaltenen Daten von vornherein „vektororientiert" und sollen leicht in die Darstellungsbasis aufgenommen werden können.

Viele geographische Informationssysteme verstehen sich als reine „Auswertungsprogramme" und sind für die Erfassung von Daten eher ungeeignet bzw. der Anschluß von GPS-Empfängern oder gar integriertes GPS Post-Processing nicht vorgesehen.

Leistungsmerkmale des Gesamtsystems:

- **Einlesen der Positionsdaten von GPS-Empfängern**
  - Direkter Anschluß aller GPS-Empfänger mit Standard NMEA-Ausgang oder Motorola Binary Format an eine serielle Schnittstelle des PC's.
  - Verbindunsaufbau via Modem/Funkmodem oder GSM zu einem Empfänger.
- **Protokollierung der GPS-Daten pro GPS-Empfänger**
  Aufzeichnung und Wiedergabe von gefahrenen Strecken mit Darstellung des zurückgelegten Weges (Track) je Fahrzeug (Empfänger ≈ Fahrzeug).
- **GPS-Statusanzeige:**
  Anzeige aller GPS-relevanten Informationen: Geographische Länge und Breite, Höhe, aktuelle Zeit und Datum, Richtung, Geschwindigkeit, Satellitenstatus (Anzahl d. empfangenen Satelliten).
- **Darstellung digitalisierter Landkarten/Satellitenbilder:**
  Kartendarstellung in verschiedenen Zoomstufen. Es können sowohl digitalisierte (beliebige Projektion), als auch Satellitenbilder eingesetzt werden.
- **Einbindung vektororientierter Daten:**
  Darstellung verschiedener vektororientierter Daten (Landes- oder Grundstücksgrenzen, Straßen, Leitungssysteme, ...). Die einzelnen Layer sollen wahlweise angezeigt werden können – mit oder auch ohne darunterliegender digitalisierter Landkarte/bzw. Satellitenbildern (Datenformat: AutoCAD DXF)

- **Integration einer Ortsdatenbank**:
  Auf der digitalisierten Karte nicht eingetragene Orte oder andere Symbole (auch vom Anwender frei definierbare) sollen nachträglich aufgrund der vom GPS-Empfänger erhaltenen Positionsdaten eingefügt werden können.

- **Manuelles Einfügen von Objekten:**
  Einfügen von Fixpunkten, Kreisen, Linien, Flächen, Polygonzügen, etc. sowie Beschriftungen mit (a) Übernahmemöglichkeit der gerade aktuellen GPS-Position und (b) Einlesen aufgezeichneter GPS-Daten vom GeoTracker[7]. Editierungsmöglichkeit für weitere Objekt-Attribute.

- **Postprocessing und Differential GPS (DGPS):**
  Die GPS-Daten der so eingegebenen Objekte sollen durch nachträgliches Bearbeiten (im Post-Processing-Verfahren) korrigiert werden können um (je nach Qualität des Empfanges) Dezimeter-Genauigkeit zu erreichen. Dazu sind zusätzlich Positionsdaten und Korrekturinformationen von einer Basisstation mit genau bekannter Position erforderlich. Diese Korrekturinformationen können von einem direkt angeschlossenen GPS-Empfänger stammen, oder werden aus einer Datei eingelesen.

- **Datumstransformation**
  Datumstransformation zur Umrechnung von WGS84-Koordinaten in lokale Systeme (das ist i.a. das gerade verwendet Kartenmaterial). Dabei kommt die Transformations-DLL eines Fremdherstellers zum Einsatz.

- **Parallele Aufzeichnung von Meßdaten:**
  Parallel zur Aufzeichnung der Positonsdaten sollen weitere Meßdaten (etwa Radioaktivität, CO_2-Werte, etc.) eingelesen werden und geeignet dargestellt werden können: z.B. durch entsprechende Verfärbung der gefahrenen Route.

- **Fahrzeugverwaltung/Datenauswertung:**
  GPS-Daten und eingelesene Meßwerte sollen je Fahrzeug entsprechend ausgewertet und dargestellt werden können.

- **Unterstützung mehrer Sprachen**
  GeoLocator ist in Englisch und Deutsch verfügbar.

[7] näheres dazu siehe: http://www.ivnet.co.at/gps/

Umgebung/Zielsystem

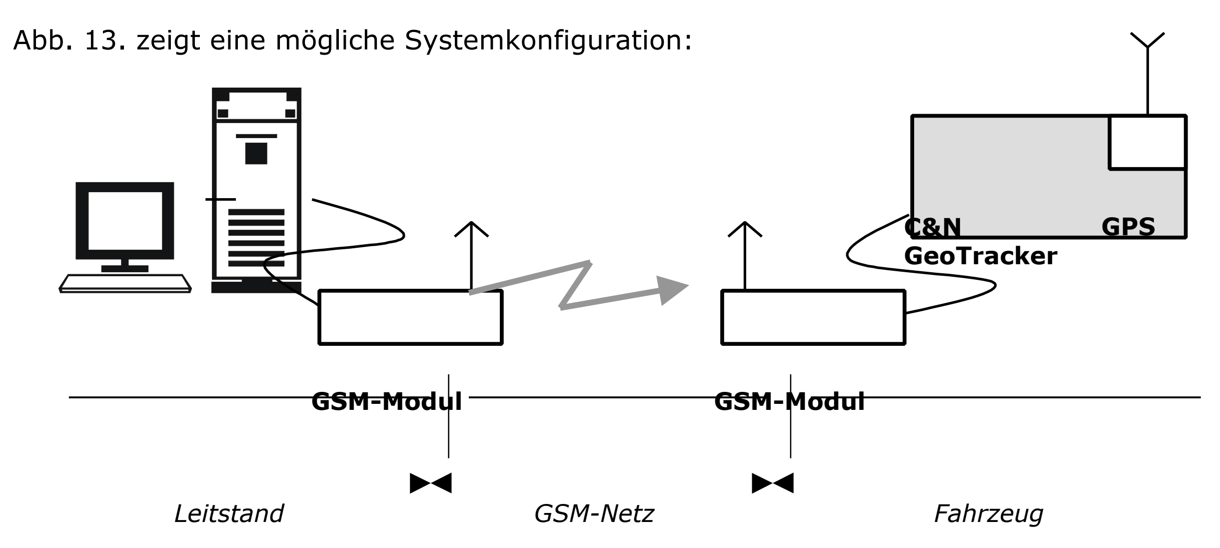
Das Hauptprogramm und Komponenten des Systems (also GeoLocator.EXE, MapAPI.DLL, GpsAPI.DLL, sowie die GPS-Treiber NMEA.GPS, Oncore.GPS und Target.GPS) wurden ausschließlich mit der Entwicklungsumgebung Borland C++ Version 5.01 und Borlands[8] OWL (Object Windows Library) erstellt.

Die Anwendung wurde für das 32Bit Microsoft Windowssystem konzipiert und entwickelt. Zum Betrieb sind also Windows 95, Window NT 4.0 oder höhere Versionen Voraussetzung. Empfohlene Hardware ist 100 MHz Pentium mit mind. 32 MB Arbeitsspeicher und CD-ROM Laufwerk.

Zusätzlich ist ein GPS-Empfänger erforderlich, der die GPS-Daten entweder im NMEA oder im Motorola Binary Format auszugeben in der Lage ist.

Um Fernortungen durchführen zu können, muß der PC mit einem SMS fähigen GSM-Gerät[9] ausgestattet sein, das über eine serielle Schnittstelle mit dem PC verbunden wird. Das zu ortende Fahrzeug muß mit einem *GeoTracker* von Communication & Navigation ausgestattet sein. Nur dieses Gerät, das zusätzlich zur GPS-Empfangseinheit mit einem eigenen Mikroprozessor ausgestattet ist, ist in der Lage die vom GeoLocator gesendeten SMS-Nachrichten entsprechend zu verarbeiten.

Abb. 13. zeigt eine mögliche Systemkonfiguration:



[8] Neuer Firmenname: *Inprise*

[9] Das GSM-Gerät muß dem ETSI Standard GSM 07.07 (ETS 300916) entsprechen. Achtung: manche Hersteller von GSM-Geräten verwenden zum Teil eigene, proprietäre Protokolle.

Systemkomponenten

Das Gesamtsystem GeoLocator gliedert sich im Wesentlichen in drei Teilsysteme: Die Anwendung GeoLocator als solche und zwei Dynamic Link Libraries, GPSAPI.DLL und der MAPAPI.DLL. Die folgende Grafik zeigt schematisch den strukturellen Aufbau:

Anwendung:
Benutzeroberfläche, Fahrzeugverwaltung

GPS-API:
Kontrolle serielle Schnittstelle, Geräteverwaltung, Koordinatentransformation, Postprocessing, ...

MAP-API:
Kartendarstellung, Zoomen, Ein-/Ausblenden von Layern und Objekten, ...

Geräte-Treiber:
Kommunikation mit Gerät, Decodierung der Signale, Konvertierung der Daten in standardisierte Strukturen

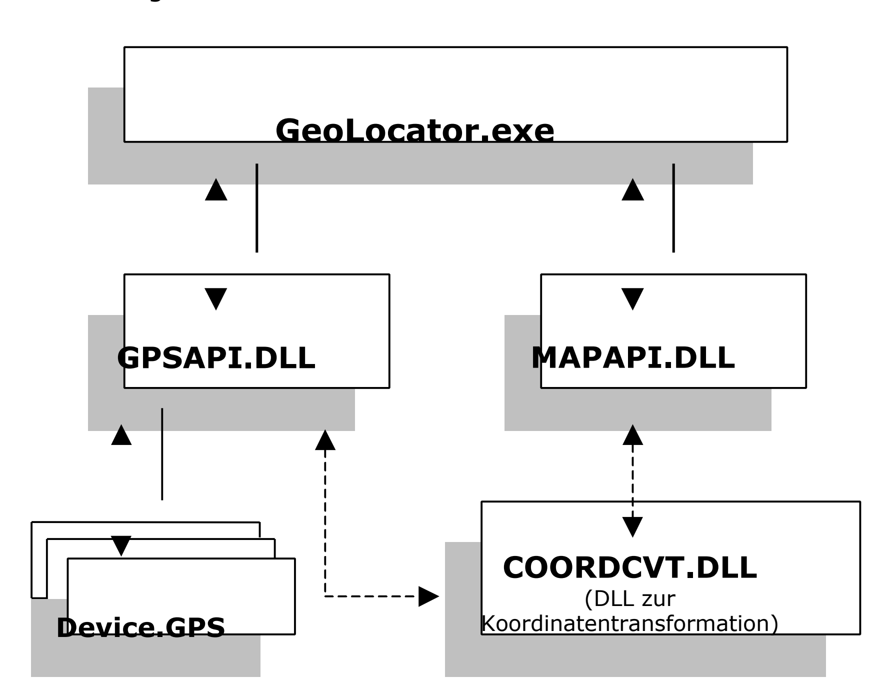


Abb. 14. Struktureller Aufbau des Gesamtsystems

GeoLocator

Der GeoLocator bildet den Rahmen für das GPSAPI und das MAPAPI. Die meisten Dialoge und Fenster sind bereits Bestandteile der genannten API's, die lediglich vom GeoLocator aufgerufen werden. Als eigene Komponenten enthält der GeoLocator eine Fahrzeugverwaltung und eine Ereignisanzeige, in der laufend eintreffende GPS-Positionen, Statusmeldungen von Geräten und andere Ereignisse mitprotokolliert werden. Zusätzlich stellt der GeoLocator Dialoge zum Lesen und Senden von Kurznachrichten (*Short Messages*) bereit, die allerdings nur dann sinnvoll eingesetzt werden können, wenn ein entsprchendes GSM-Gerät vorhanden ist.

Eine genaue Ausführung folgt etwas weiter unten in diesem Kapitel.

GPS Application Programming Interface

Das *GPS Application Programming Interface* stellt für Anwendungsprogramme eine einheitliche Schnittstelle und einheitliche Datenstrukturen für GPS-Geräte unterschiedlicher Hersteller bereit. Für neue GPS-Geräte muß dazu eine spezifische Geräte-DLL mit exakt definierter Schnittstelle erstellt werden, die allerdings nur ganz wenige essentielle Decodierungsroutinen und Setupdialoge enthalten muß. Der gesamte Aufwand der Kommunikation über die serielle Schnittstelle wird bereits im Kern des GPSAPIs abgewickelt.

Eine Beschreibung des GPSAPIs und welche Aufgaben die einzelnen Gerätetreiber übernehmen müssen ist detailliert in Kapitel 4 beschrieben.

Map Application Programming Interface

Dem MapAPI kommen alle Aufgaben eines geografischen Informationssystems zu: Darstellung von Karten, GPS-Positionen und Routen, Scrollen, Zoomen, Messen von Distanzen zwischen bestimmten Positionen auf der Karte, usw.

Innerhalb einer Karte (Map) wird jedes darzustellende Objekt als sog. *Layer* bezeichnet. Es kann sich dabei um ein Bitmap (z.B. eine gescannte Landkarte), um vektororienterte Daten (z.B. eine zurückgelegte Strecke) oder manuell eingetragenen Punkte (vorgegebene Vermessungspunkte) handeln. Mehrere Layer können in *Gruppen* zusammengefaßt werden (z.B. eine Gruppe von Fahrzeugen).

In der Reihenfolge, in der Layer in eine Karte eingefügt werden, werden sie später auch dargestellt: Später eingefügte Layer kommen somit auf einer „höheren Ebene“ zu liegen und können daher darunterliegende überdecken.

Kapitel 5 widmet sich dem MapAPI im Detail.

Koordinatentransformation

Eine wesentliche Aufgabe des GeoLocators ist die Darstellung der vom GPSAPI erhaltenen dreidimensionalen Koordinaten in der Ebene. Dazu ist es erforderlich die von GPS-Empfängern standardmäßig im WGS84[10] gelieferten Positionen in das jeweils verwendete Landeskoordinatensystem umzuwandeln.

Es besteht die Möglichkeit entweder die Koordinaten bereits bei der Erfassung in ein bestimmtes Landeskoordinatensystem zu transformieren oder jeweils bei der Darstellung. Mit dem GpsAPI und dem MapAPI sind beide Möglichkeiten realisierbar, da beide DLLs Prozeduren zur Koordinatentransformation bereitstellen.

Der GeoLocator macht von der zweiten Möglichkeit Gebrauch: Alle Einstellungen bzgl. des Koordinatensystems beziehen sich auf die aktuelle, vom MapAPI geöffnete Karte. Es ist daher ohne weiteres möglich, unterschiedliches Kartenmaterial gleichzeitig zu verwenden. Grundsätzlich können die Koordinaten immer wieder neu transformiert werden, da die im GPSAPI definierte und systemweit verwendete Datenstruktur für GPS-Positionen die originalen WGS84 Koordinaten enthält.

Sowohl GpsAPI als auch MapAPI verwenden zur Koordinatentransformation eine gekaufte Danymic Link Library eines Fremdherstellers. Dieses Bibliothek ist in der Lage, Koordinaten zwischen beliebigen Koordinatensystemen umzurechnen. Die meisten gängigen Koordinatensysteme sind bereits vordefiniert. Zusätzlich bietet die Bibliothek aber auch die Möglichkeit, neue Koordinatensysteme zu definieren.

[10] World Geodetic System 1984

Datenflußdiagramm

Die folgende Skizze versucht den Zusammenhang zwischen GPSAPI, GeoLocator, MapAPI und Koordinatentransformations-DLL (CoordCvt.dll) zu verdeutlichen, sowie den Datenfluß vom GPS-Gerät bis zum MapAPI zu veranschaulichen:

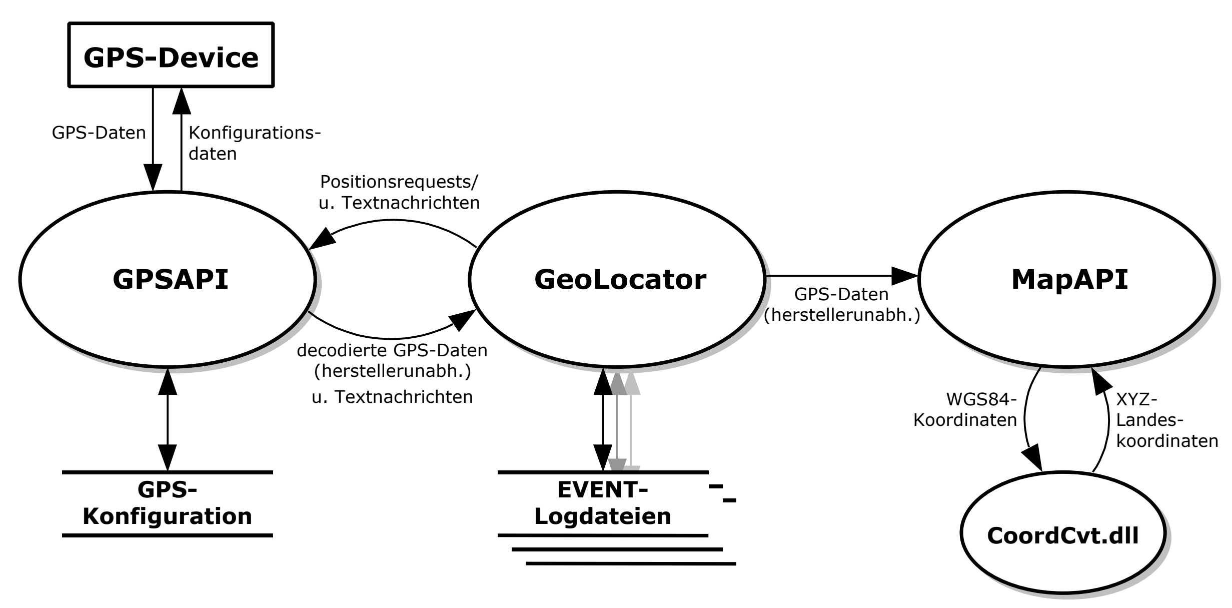


Abb. 15. Datenflußdiagramm GPS-Daten

Beim Programmstart öffnet der GeoLocator über das GPSAPI eine Verbindung zu allen angeschlossenen GPS-Geräten. Das GPSAPI initialisiert dabei die Geräte indem es die jeweiligen Konfigurationsdaten an das Gerät sendet. Die gesamte GPS-Konfiguration, d.h. welche Geräte überhaupt angeschlossen sind und wie sie konfiguriert sind, wird vom GPSAPI in der Registry des lokalen Rechners gespeichert. Zum Bearbeiten stellt das API zusammen mit den GPS Gerätetreibern entsprechende Dialoge zur Verfügung.

Die von den GPS-Geräten angelieferten GPS-Daten (im herstellerspezifischen Format) werden vom GPSAPI mit Hilfe der GPS-Gerätetreiber decodiert und in ein herstellerunabhängiges Format gebracht. In dieser Form wird die GPS-Position an den GeoLocator geliefert. Genaugenommen wird nur eine Windows-Message an den GeoLocator gesendet, der daraufhin die eine bestimmte GPSAPI-Funktion aufruft und sich so die Daten holt.

Handelt es sich bei der erhaltenen GPS-Position um ein Fahrzeug, das auf der gerade geöffneten Karte dargestellt werden soll, so wird die Position an das MapAPI weitergereicht. Das MapAPI rechnet die im WGS84 vorliegenden Koordinaten in das zur Karte passende Landeskoordinatensystem um. Diese Aufgabe wird von der Transformationsbibliothek CoordCvt.dll erledigt.

Zusätzlich werden vom GeoLocator alle GPS-Positionen (und auch andere Ereignisse, wie Statusmeldungen oder Textnachrichten) in ein Event-Logfile geschrieben. Für jeden Typ von Ereignis existiert ein eigenes Logfile.

Objektmodell

Das folgende Objektmodell[11] dient in erster Linie dazu, einen Überblick über den GeoLocator zu geben. Datenelemente und Methoden der einzelnen Objekte sind daher nur ansatzweise skizziert. Bezüglich Details sei auf die im Anschluß angegebenen Objektbeschreibungen bzw. die entsprechenden Headerdateien verwiesen.

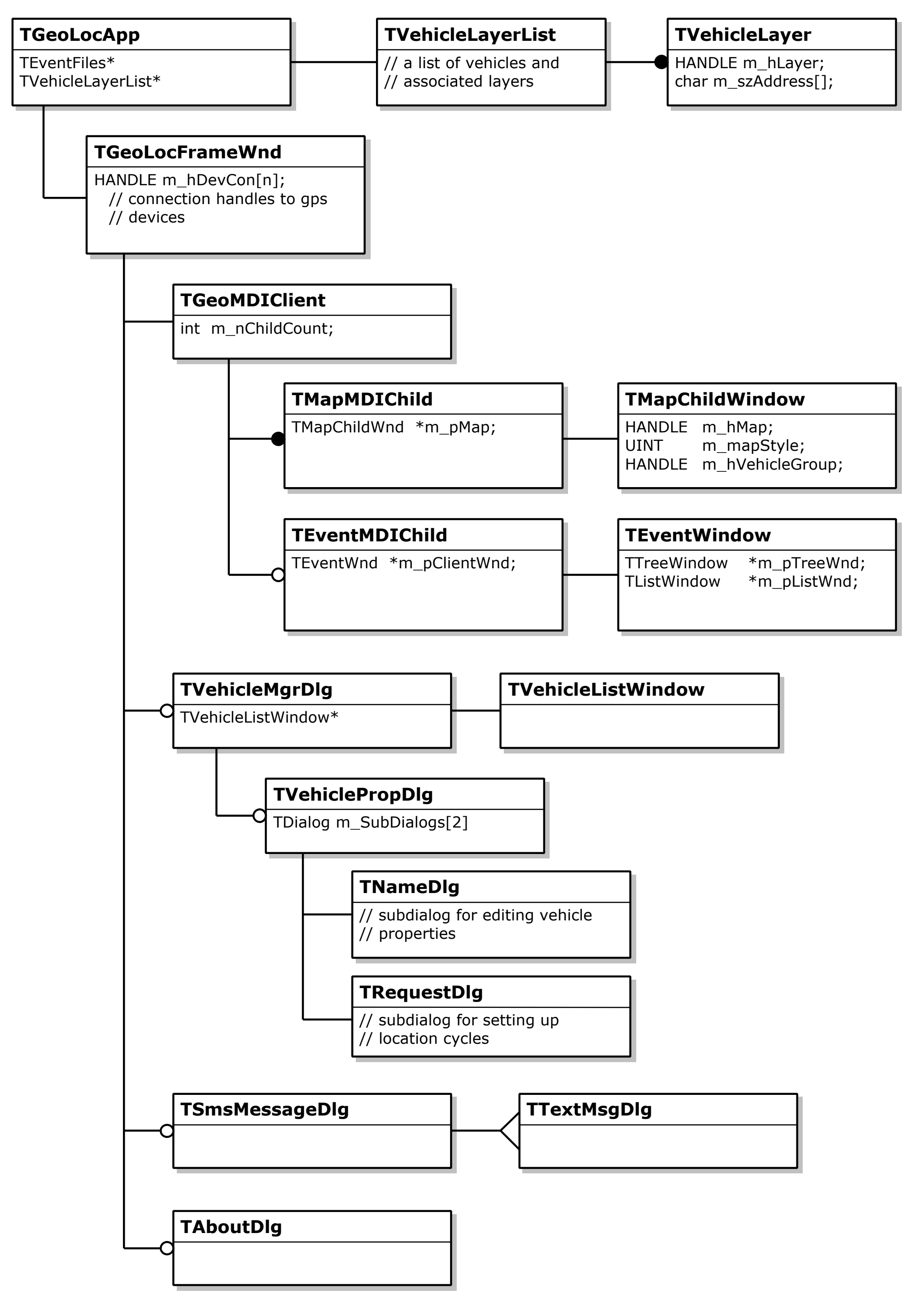


Abb. 16. Objektmodell GeoLocator

11 Die verwendete Notation orientiert sich an: Rumbaugh, „Object Oriented Modelling and Design", Prentice Hall, 91

Klasse TGeoLocApp

Die Klasse TGeoLocApp, das direkt vom OWL-Objekt TApplication abgeleitet ist, enthält neben den ererbten Funktionen zur Behandlung von Windows-Nachrichten und Benutzerkommandos auch alle Funktionen für den Zugriff auf die Ereignisdateien. Zusätzlich kapselt es auch alle DLL-Funktionsaufrufe des GPSAPI's und des MapAPI's. Alle Rückmeldungen vom GPSAPI werden ebenfalls an das Hauptmodul gesendet.

Klasse TGeoLocFrameWnd

ist das Rahmenfenster für die Anwendung. Dieses Fenster baut eine Verbindung zu allen vorhandenen GPS-Geräten auf. Vice versa schließt es alle Verbindungen bei Programmende.

Klasse TGeoMDIClient

ist das MDI Client-Fenster der Anwendung und für die Verwaltung aller MDI-Child-Fenster verantwortlich.

Klasse TMapMDIChild

Innerhalb dieser MDI-Child-Fenster werden die Kartenansichten geöffnet. Von dieser Klasse von MDI-Fenstern können (theoretisch) beliebig viele Instanzen erzeugt werden.

Klasse TMapChildWindow

Fensterklasse für den Client-Bereich von `TMapMDIChild`. Innerhalb dieses Fensters wird mit dem Funktionsaufruf von `mapapi_OpenMap()` die eigentliche Karte angelegt.

Klasse TEventMDIChild

Ist das Fenster für die Ereignisanzeige. Die Ereignisanzeige kann nur ein einziges Mal geöffnet werden.

Klasse TEventWindow

Fensterklasse für den Client-Bereich von `TEventMDIChild`. Dieses Fenster ist in zwei Hälften geteilt: Auf der linken Seite befindet sich eine Baumansicht zum Auswählen des Ereignistypes, rechts davon die Ereignisliste.

Klassen TSmsMessageDlg und TTextMsgDlg

Mit dem Dialog TTextMsgDlg können Kurznachrichten versandt, gelesen bzw. beantwortet werden. Die Basisklasse TSmsMessageDlg führt bestimmte Überprüfungen (z.B. Validierung der Mobilrufnummer) durch.

Klassen TVehicleLayer und TVehicleLayerList

In dieser dynamischen Liste von `TVehicleLayer`-Objekten verwaltet der GeoLocator alle Fahrzeuge, denen ein Layer in einer Karte zugeordnet ist. Trifft eine neue GPS-Position ein, wird diese Liste nach vorhanden Fahrzeug-Layern durchsucht, und die neue Position an das MapAPI zum jeweiligen Layer weitergeleitet.

Klasse TVehicleMgrDlg
Klasse TVehicleListWindow

Dem `TVehicleMgrDlg` obliegt die Fahrzeugverwaltung. Alle vorhandenen Fahrzeuge werden im `TVehicleListWindow` aufgelisetet. Fahrzeugdaten und Ortungszyklen können mit diesen beiden Dialogen festgelegt werden.

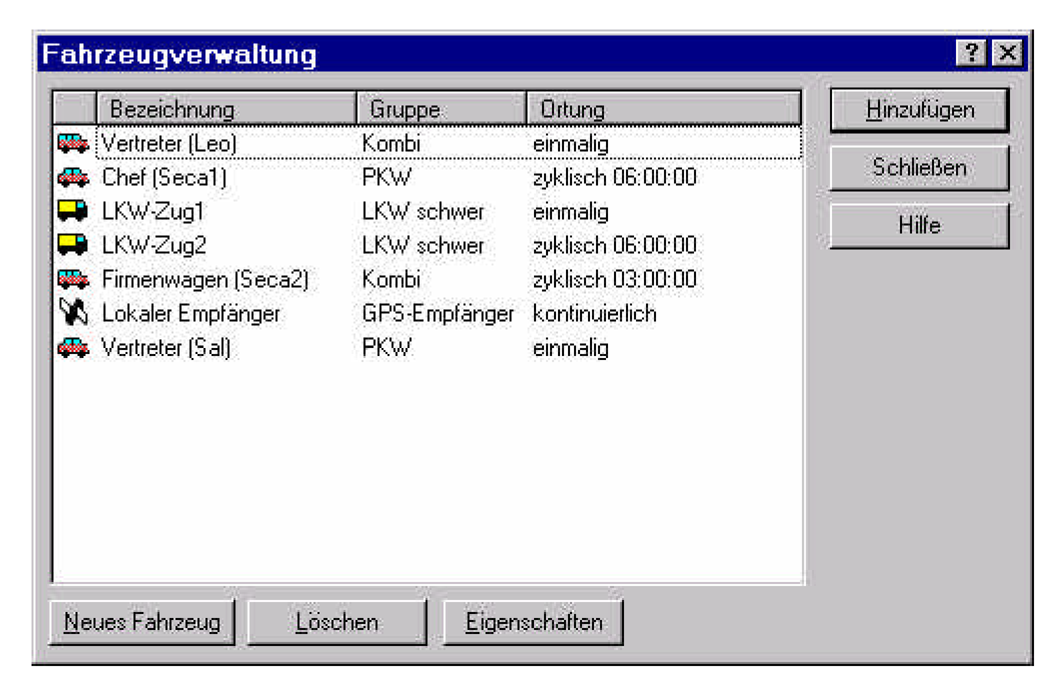


Abb. 17. Ansicht von TVehicleMgrDlg

Klasse TVehiclePropDlg
Klasse TNameDlg
Klasse TRequestDlg

Im Dialog `TVehiclePropDlg` können alle Fahrzeugdaten eingegeben und Ortungseinstellungen vorgenommen werden. Der Dialog besteht aus zwei Subdialogen: dem `TNameDlg`, in dem u.a. Fahrzeugbezeichnung, Identifikationsnummer und zugordnetes GPS-Gerät eingestellt werden müssen und dem `TRequestDlg`, in dem festgelegt werden kann ob, wann und wie oft eine Position vom Fahrzeug angefordert werden soll.

Im eingangs skizzierten Objektmodell des GeoLocators nicht eingetragen sind die beiden folgenden Dialogklassen:

Klasse TPhoneBookDlg
Klasse TRecipientDlg

Diese beiden Dialogklassen dienen zum Bearbeiten des internen Telefonbuches (Phonebk.dat). Es handelt sich bei dieser Datei um eine einfache Textdatei, die mit den Windowsfunktionen zum Bearbeiten von Profiles verwaltet wird. Der TPhoneBookDlg wird vom Dialog TSmsMessageDlg aufgerufen.

Fahrzeugverwaltung

Die Fahrzeugverwaltung als solche befindet sich im Dialog TVehicleMgrDlg. Von hier aus können alle Fahrzeugdaten konfiguriert werden. Die Fahrzeuge selbst können über Funktionen der dynamische Liste TVehicleList hinzugefügt, gelöscht, oder Einstellungen geändert werden.

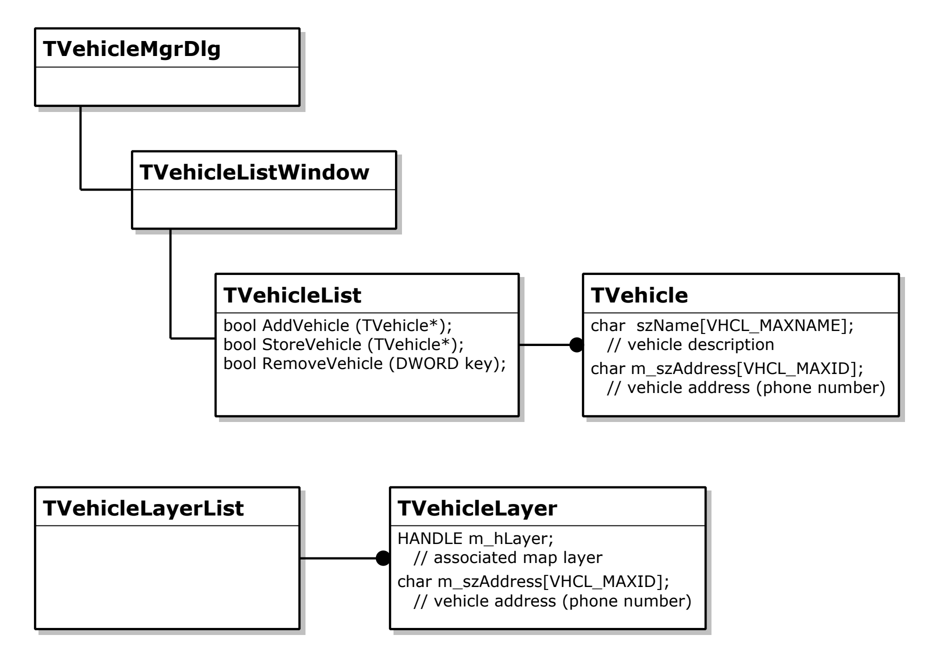


Abb. 18. Objektmodell Fahrzeugverwaltung

Da die Fahrzeugverwaltung nur für eine relativ kleine Anzahl von Fahrzeugen ausgelegt ist (n < 1000), werden die Daten von TVehicleList in die Textdatei Vehicles.dat kurzerhand mit den Windows-Profilefunktionen (WritePrivateProfileString, GetPrivateProfileString, ...) gespeichert bzw. gelesen. Ein Eintrag für ein Fahrzeug sieht wie folgt aus:

```
[00000007]
Name=VW Passat Variant
No=UU-73TM
Id=+436764759093
Group=1
GpsDevice=1
LocateParams=70E70300002C0100006054000000E1000000001C
```

Für jedes Fahrzeug wird eine eigene *Section* angelegt. Wichtig sind die Einträge Id und GpsDevice. GpsDevice identifiziert das Gerät, über das das Fahrzeug erreichbar ist. Es handelt sich dabei um den Index auf die am System angeschlossenen Geräte. 1 bedeutet im konkreten Fall also, daß das Fahrzeug über das erste an den PC angeschlossene GPS-Gerät erreichbar ist.

Id ist die Adresse des Fahrzeuges. Im oben angeführten Beispiel handelt es sich offensichtlich um eine Mobilrufnummer – beim angeschlossenen GPS-Gerät handelt es sich daher um ein GSM-Modul. Rufnummern müssen übrigens immer im Internationalen Format (+43...) angegeben werden.

Hinter LocateParams verbirgt sich eine Datenstruktur (GPSAPI_LOCATEPARAMS), die alle Einstellungen bzgl. der Ortung (einmalige Ortung, zyklisches Intervall, etc.) beinhaltet. Wird vom Fahrzeug nun eine Position angefordert, so wird diese Datenstruktur an den *C&N*

GeoTracker, der im Fahrzeug installiert sein muß, gesendet. `GPSAPI_LOCATEPARAMS` wird vom GpsAPI definiert, und wird im entsprechenden Kapitel noch näher beschrieben.

Zuordnung Position eines Fahrzeug zu Layer auf der Karte

Für jedes Fahrzeug kann auf einer oderer mehreren Karten (Maps) eine sog. *Layer* eingerichtet werden. Um eine eingetroffene GPS-Position dem richtigen Layer zuordnen zu können, verwendet der GeoLocator die dynamische List `TVehicleLayerList`. In deren Einträgen TVehicleLayer wird jeweils das Layer-Handle (das vom MapAPI beim Anlgen eines neuen Layers zurückgegeben wird und das wir zum Einfügen von Positionen in einen Layer benötigen) und die Adresse (das ist i.a. eine Telefonnummer) des Fahrzeuges gespeichert.

Trifft nun eine GPS-Position von einem Fahrzeug ein, durchsucht der GeoLocator die dyamische Liste nach TVehicleLayer-Objekten, die dieselbe Adresse (Telefonnummer) besitzen und fügt die Position zum gefundenen Layer per MapAPI-Funktionsaufruf hinzu.

Ereignisverwaltung

Jedes vom GPSAPI eintreffende Ereignis, sei es eine GPS-Position, eine Gerätestatusmeldung oder eine Textnachricht wird in einer Ereignisdatei abgelegt. Mithilfe der Ereignisanzeige kann man in die jeweilgen Ereignisdatei Einsicht nehmen.

Positionsanforderungen und zu sendende Textnachrichten (diese werden vom GeoLocator selbst generiert) werden genauso wie eingehende Ereignisse behandelt.

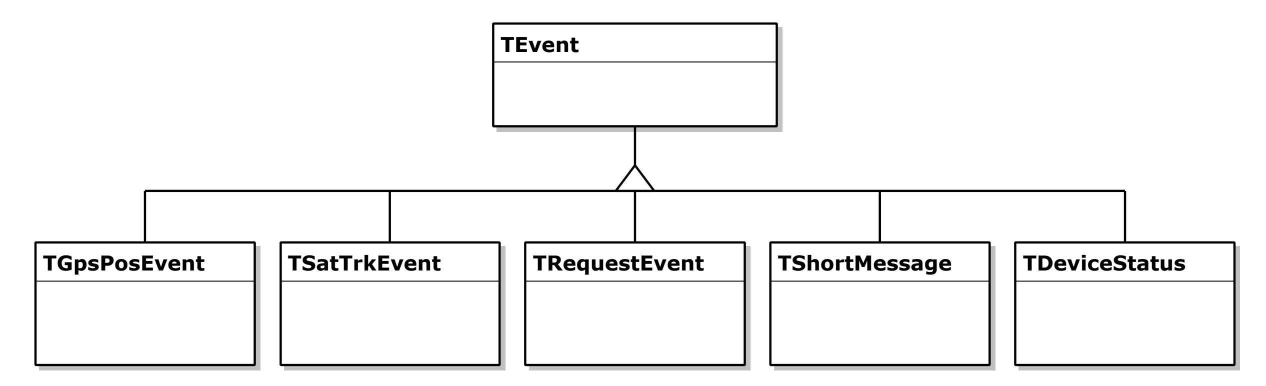


Abb. 19. Objektmodell Ereignisse

Ereignistypen

Alle Ereignistypen werden von der virtuellen Basisklasse `TEvent` abgeleitet, die damit die Schnittstelle für alle Ereignisobjekte festlegt. In der aktuellen Implementierung gibt es fünf verschiedene Typen von Ereignissen:

- **Ereignis GPS-Position (*TGpsPosEvent*)**

  In der GPS-Positionsnachricht sind enthalten: Länge, Breite, Höhe in WGS84 Koordinaten, Datum und Uhrzeit bezüglich UTC[12], Geschwindigkeit, Richtung, DOP-Wert, sowie Anzahl der sichtbaren und Anzahl der „verfolgten" Satelliten.

- **Ereignis Satellitenzustand bzw. Trackingstatus (*TSatTrkEvent*)**

  In der Trackingstatusnachricht sind Nummer des Satelliten, Tracking Modus, und Signalstärke eines jeden der momentan sichtbaren Satelliten enthalten.

- **Ereignis Kurznachricht (*TShortMessage*):**

  Sofern eine GSM-Gerät an den PC angeschlossen ist, werden ein- und ausgehende Kurznachrichten in dieser Datenstruktur gespeichert.

- **Ereignis Positionsanforderung (*TRequestEvent*)**

  Ist ein Empfänger direkt an den PC angeschlossen, so sendet er je nach Geräteeinstellungen) seine Positionen unaufgefordert an den PC. Soll jedoch die Positionen eines entfernten Fahrzeuges über ein GSM-Gerät ermittelt werden, so müssen dessen Positionsdaten angefordert werden. Solche Positionsanforderungen werden in dieser Datenstruktur gespeichert.

- **Ereignis Gerätezustand (*TDeviceStatus*)**

  Bei jeder Änderung des Gerätezustandes eines GPS-Gerätes (z.B. *busy*, *idle*) sendet das GPSAPI eine spezielle Nachricht an die Anwendung, für die dieses Ereignisobjekt verwendet wird.

Jedes der Ereignisse wird in einer separaten Datei (Dateiendung .dat) abgelegt. Der Aufbau der Datei wird durch das jeweilige Ereignisobjekt bestimmt. Alle notwendigen Grundoperationen, wie Einfügen, Löschen, wahlfreier Zugriff auf bestimmte Datensätze werden durch ein Template festgelegt. Mit Hilfe dieses Templates und der jeweiligen Ereignisklasse wird für jedes Ereignis das entsprechende Dateiobjekt deklariert.

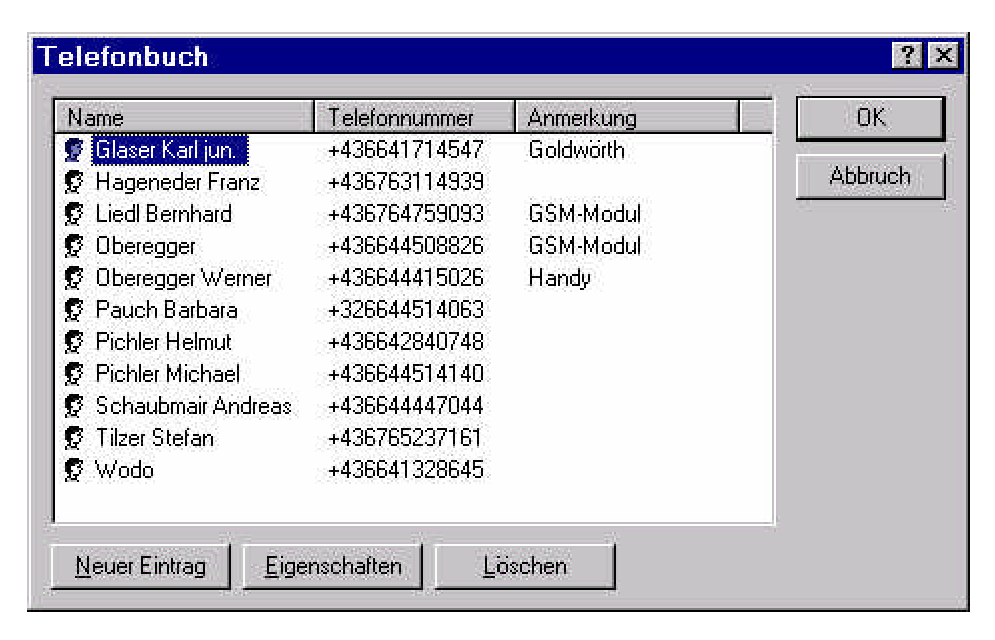
Der GeoLocator kann daher leicht um neue Ereignise erweitert werden. Es genügt ein neues Objekt von TEvent abzuleiten und mit dem Ereignisdatei-Template eine neue Datei zu definieren.

Aufgrund der großen Datenmengen die anfallen können (GPS-Empfänger geben üblicherweise Positionen im Sekundentakt aus) kann es für andere Anwendungen aber sehr wohl sinnvoll sein, auf eine Datenbank zur Ablage von Positionsdaten zurückzugreifen.

[12] UTC ... Universal Time Coordinated.

Telefonbuch

Innerhalb der Ereignisverwaltung ist auch die Möglichkeit zum Lesen und Versenden von Short Messages vorhanden. Damit nicht jedesmal beim Verfassen einer Nachricht die Nummer erneut eingetippt werden muß, kann man sie aus dem Telefonbuch auswählen.



Dateireferenz

Verzeichnisstruktur

Das gesamte Projekt GeoLocator (einschließlich MapAPI und GPSAPI) umfaßt folgende Verzeichnisse:

\exe	Executables, enthält die Anwendung GeoLocator.exe selbst und alle zum Betrieb notwendigen DLL's: MapAPI.dll, GPSAPI.dll, NMEA.gps, Oncore.gps, Target.gps, CoordCvt.dll (Koordinaten-transformation)
\exe\data	Verzeichnis für Fahrzeugverwaltung (Vehicles.dat), Telefonbuch (PhoneBk.dat) und Ereignisdateien (Requests.dat, GpsPos.dat, SMSin.dat, SMSout.dat)
\Doku	Projektdokumentation, Gerätespezifikationen, Datenblätter, ...
\GeoLocator	Verzeichnis für Projektdateien, Sourcecodes und Resourcen
\GeoLocator\Doku	Dokumentationen nur den GeoLocator betreffend, Bugreport
\GeoLocator\Source	Sourcecode des GeoLocators (.h und .cpp Dateien)
\GeoLocator\Res	Resourcen (.rc/.rh Dateien, Bitmaps, ...)
\GeoLocator\Obj	Objectcodes GeoLocator
\GpsAPI	Sourcecode und Resourcen für das GPSAPI (Unterverzeichnisse analog zu GeoLocator)
\GpsDevs	Sourcecode und Resourcen aller GPS-Gerätetreiber (Unterverzeichnisse analog zu GeoLocator)

\MapAPI	Sourcecode und Resourcen für das MapAPI (Unterverzeichnisse analog zu GeoLocator)
\Misc\Source	Sourcecode und Resourcen die von mehreren oder allen Teilprojekten benötigt wird.
\CoordCvt	Header-Dateien und zum linken notwendigen Dateien der Koordinatentransformations-DLL, sowie die Dokumentation.
\Archiv	Sourcecode älterer Programmversionen

Sourcecodes GeoLocator

In der folgenden Tabelle sind alle C++ Source- und Header-Dateien aufgelistet, die zum Compilieren des Hauptprogrammes `GeoLocator.exe` erforderlich sind.

Datei	**Beschreibung**
Verzeichnis *GeoLocator\Source:*	
GeoLoc.h/.cpp	Hauptmodul. Definition/Implementierung von TGeoLocApp
GeoMDIC.h/.cpp	MDI Client-Fenster. Rahmen für alle MDI-Child-Fenster
MapChild.h/.cpp	MDI-Child-Fensterobjekt für die Kartendarstellung
Vehicles.h/.cpp	Definition/Implementierung von TVehicle und TVehicleList
DVhclMgr.h/.cpp	Dialogobjekt für die Fahrzeugverwaltung
DVhclPrp.h/.cpp	Dialogobjekt zum Bearbeiten der Fahrzeugoptionen
EvtChild.h/.cpp	MDI-Child-Fensterobjekt für die Ereignisanzeige
Event.h/.cpp	Definition/Implementierung der Ereignisobjekte
MsgDlg.h/.cpp	Dialogobjekt zum Erstellen und Bearbeiten von Textnachrichten
DPhoneBk.h/.cpp	Dialogobjekt zum Verwalten der Telefonbucheinträge
DRecp.h/.cpp	Dialogobjekt zum Bearbeiten von Telefonbucheinträgen
DAbout.h/.cpp	Info-Dialogobjekt
GeoLoc.def	Projekt-Definitionsdatei
Verzeichnis *GeoLocator\Res:*	
GeoLoc.rh	Definitionen und Konstanten für alle Resourcen
GeoLoc.rc	Beinhaltet alle sprachabhängigen Resourcen (Dialoge, Strings, ...)
Bitmaps.rc	Bitmaps, Icons, Cursors
Verzeichnis *misc\Source:*	
log.h/.cpp	Logfile-Funktionen (für Debugging Zwecke)
Verzeichnis .*exe:*	
GpsAPI.lib	Bibliothek zum Linken des GpsAPI's (Loadtime-Linking)
MapAPI.lib	Bibliothek zum Linken des MapAPI's (Loadtime-Linking)

Kapitel 4

Dokumentation GPSAPI

Dieses Kapitel erläutert Zweck und Funktionsweise des *Global Positioning System Application Programming Interfaces* (kurz GPSAPI genannt). Die GPSAPI Spezifikation definiert eine Schnittstelle für die Verwendung von GPS Empfängern unter Microsoft Windows™.

Einleitung

Zweck des GPSAPI's ist es vom jeweils angeschlossenen GPS-Empfänger abstrahieren zu können, ebenso davon ob der Empfänger direkt am Gerät angeschlossen ist oder die Verbindung über ein GSM-Modul oder eine Funkstrecke erfolgt.

Hauptaufgabe des API's ist es Positionsdaten in einem vom GPS-Empfänger unabhängigen Datenformat an die aufrufende Anwendung abzuliefern. Es exisiert mit NMEA-0183[13] zwar ein Standard für den Austausch von GPS-Positionen, dieser ist aber insofern ungeeignet für die Weiterverarbeitung, da es sich um eine reine Textrepräsentation handelt, die außerdem für bestimmte Zwecke notwendige Satelliteninformationen nicht mehr enthält. Darüberhinaus stehen z.B. Positions- und DOP-Wert gemäß NMEA Stanard in verschiedenen Datensätzen, die hintereinander übertragen werden, jedoch häufig gleichzeitig benötigt werden.

Die Hersteller von GPS-Empfängern verwenden meist eigene, binäre Repräsentationen, die ggf. auch die für ein Postprocessing notwendigen GPS-Rohdaten enthalten. Aus diesem Grund führt das GPSAPI zwei neue GPS-Datenstrukturen ein, die alle notwendigen GPS-Informationen enthalten: `GPSAPI_POS` für Position, Uhrzeit und Datum, Geschwindigkeit, etc, sowie `GPSAPI_SATTRK` für den Tracking-Status (Anzahl der Satelliten, Signalstärke, etc).

Da das GPSAPI auch für den Einsatz in der Fahrzeugortung konzipiert ist, und in diesem Bereich neben den Positionsdaten häufig auch andere Nachrichten[14] zwischen Fahrzeug und Leitstand ausgetauscht werden müssen, ist im API auch eine Schnittstelle zum Senden und Empfangen von Kurznachrichten integriert.

[13] National Marine Electronics Association
[14] In manchen Speditionsbetrieben kommuniziert der Disponent mit den Fahrern vielfach nur mehr über SMS.

Komponenten des GPSAPI

Das GPSAPI besteht aus zwei wesentlichen Komponenten: der GPSAPI.DLL selbst sowie einer Menge von GPS-Gerätetreibern (Dateiendung .GPS; es handelt sich aber ebenfalls um DLL's).

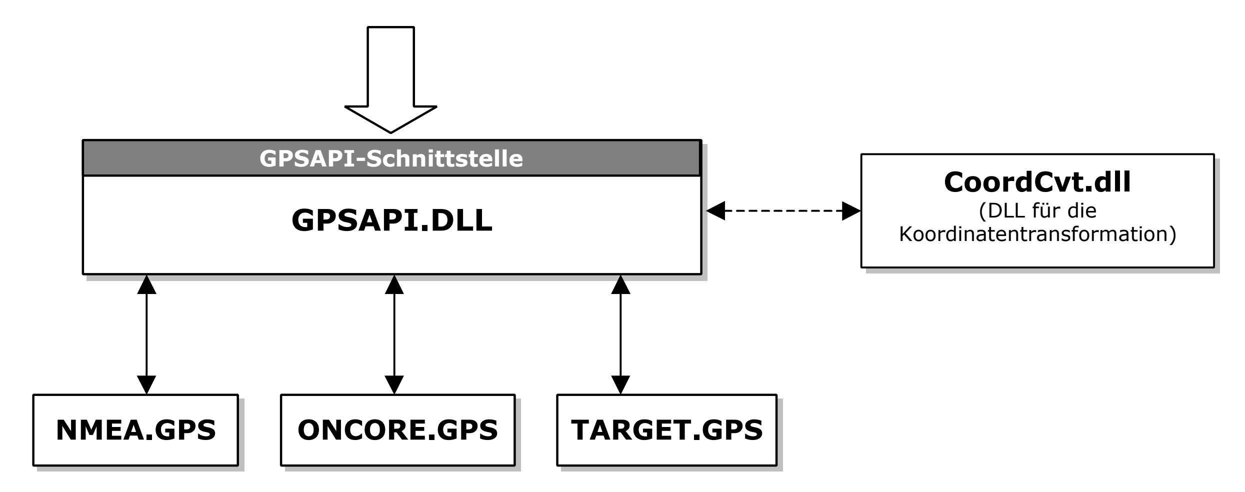


Abb. 20. GPSAPI und die zur Zeit verfügbaren GPS-Gerätetreiber

Wesentliche Aufgaben des GPSAPI sind:

- Handhabung der seriellen Schnittstellen und Kommunikation mit den GPS-Empfängern
- Zwischenspeicherung der eingelesenen Daten, bis sie vom GPS-Gerätetreiber decodiert wurden
- Zwischenspeicherung der an die Anwendung zu übergebenden GPS-Daten und Ergebnisse
- Mitführen von Protokollen für GPS-Geräte.

Ein Gerätetreiber stellt dem GPSAPI eine Decodierroutine zum Decodieren von GPS-Daten zur Verfügung und enthält darüberhinaus noch einen Setup-Dialog, um gerätespezifische Einstellungen vornehmen zu können.

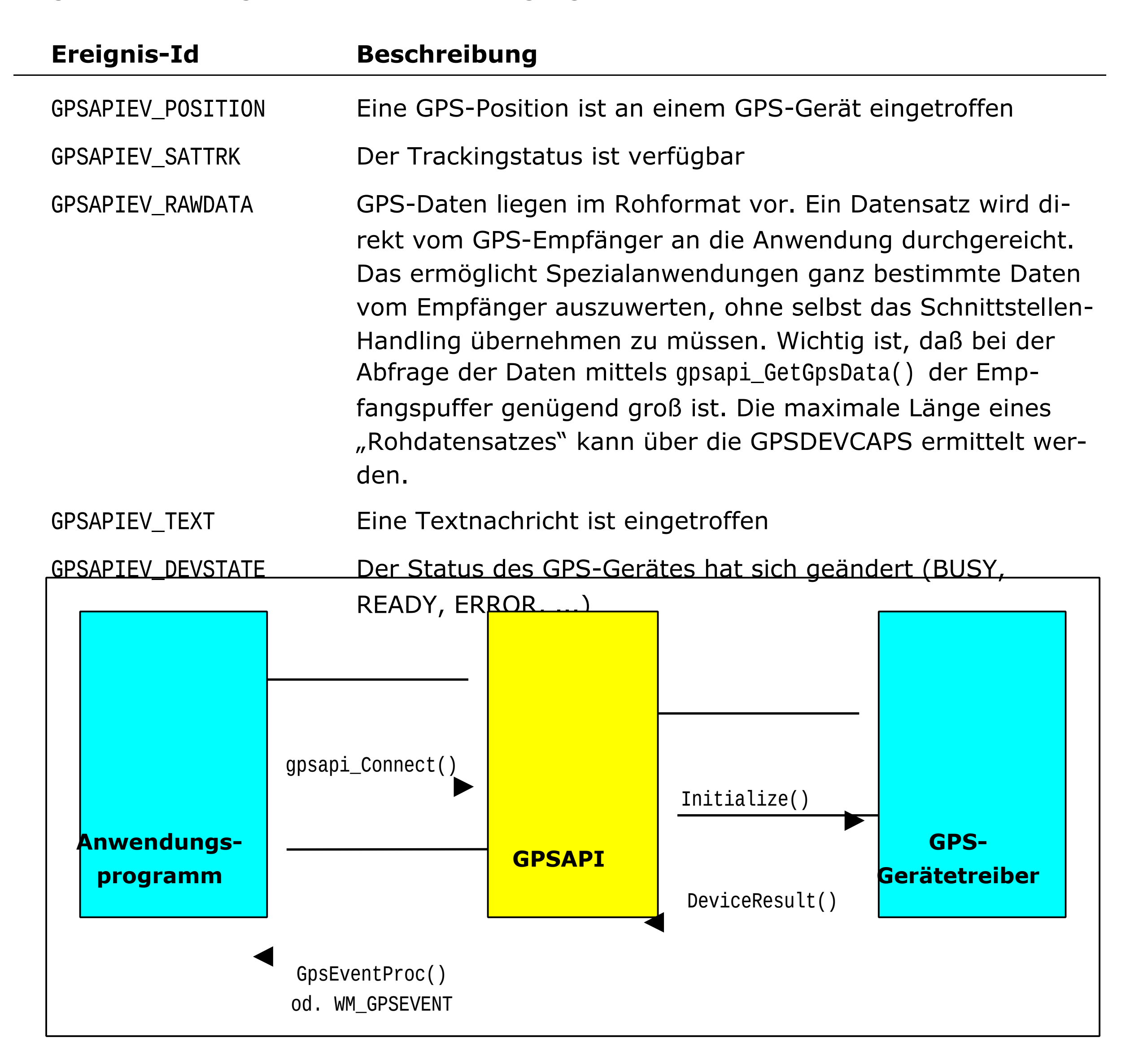
Leitfaden zur Anwendung des GPSAPI’s

Zentrale Funktion des GPSAPI’s ist die Funktion **gpsapi_Connect**. Mit dieser Funktion stellt eine Anwendung die Verbindung zu einem angeschlossenen GPS-Empfänger her. Die Anwendung wird in der Folge von GPSAPI über bestimmte Ereignisse, wie z.B. „GPS-Position eingetroffen“ benachrichtigt. Das kann auf zweierlei Arten erfolgen: entweder ruft das GPSAPI eine von der Anwendung bereitgestellte Callback-Funktion auf, oder das GPSAPI sendet eine vordefinierte Nachricht an das Hauptfenster der Anwendung.

Zusätzlich zur Art, wie das GPSAPI die Anwendung über Ereignisse benachrichtigt, wird mit dem Aufruf von gpsapi_Connect auch bereits festgelegt, welche Ereignisse eine Benachrichtigung auslösen sollen.

Folgende GPS-Ereignisse stehen zur Verfügung:

Ereignis-Id	**Beschreibung**
`GPSAPIEV_POSITION`	Eine GPS-Position ist an einem GPS-Gerät eingetroffen
`GPSAPIEV_SATTRK`	Der Trackingstatus ist verfügbar
`GPSAPIEV_RAWDATA`	GPS-Daten liegen im Rohformat vor. Ein Datensatz wird direkt vom GPS-Empfänger an die Anwendung durchgereicht. Das ermöglicht Spezialanwendungen ganz bestimmte Daten vom Empfänger auszuwerten, ohne selbst das Schnittstellen-Handling übernehmen zu müssen. Wichtig ist, daß bei der Abfrage der Daten mittels `gpsapi_GetGpsData()` der Empfangspuffer genügend groß ist. Die maximale Länge eines „Rohdatensatzes“ kann über die GPSDEVCAPS ermittelt werden.
`GPSAPIEV_TEXT`	Eine Textnachricht ist eingetroffen
`GPSAPIEV_DEVSTATE`	Der Status des GPS-Gerätes hat sich geändert (BUSY, READY, ERROR, ...)



Die Anwendung wird bei einem GPS-Ereignis durch eine Nachricht oder durch die Callback-Funktion `GpsEventProc` *benachrichtigt.*

Eine Anwendung kann gleichzeitig mehrere Verbindungen zu einem GPS-Gerät aufbauen. So ist es durchaus denkbar, daß ein Thread der Anwendung Positionen behandelt, ein weiterer Thread sich nur um Textnachrichten kümmert.

Zusammenhang zwischen GPS-Gerätetreibern, internem TDevice-Objekt und Verbindungen (TConnections)

Das GPSAPI verwaltet alle verwendeten GPS-Geräte in einer Geräteliste (TDeviceList). Jedem GPS-Gerät (TDevice) in dieser Liste ist eine Verbindungsliste (TConnectionList) zugeordnet, in der alle offenen Verbindungen (TConnections) seitens der Anwendung verwaltet werden. In diesen Verbindungsobjekten wird gespeichert, wie die Anwendung über Ereignisse benachrichtigt werden soll (Callback oder Message) und über welche Ereignisse die Anwendung benachrichtgt werden will.

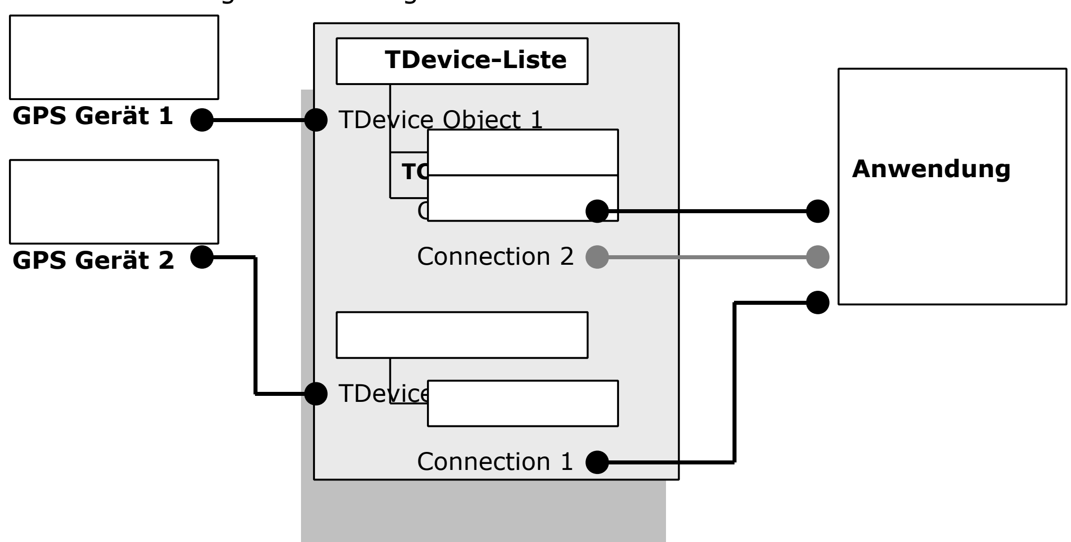


Abb. 21. Skizze der Geräteliste (TDeviceList)

Vom GPS Gerät eintreffende Ereignisse werden nun an die Verbindungsobjekte weitergereicht. Je nach Art der Verbindung gibt es nun zwei Möglichkeiten:

Fall 1: Callback-Mechanismus:

Hat die Anwendung dieses Ereignis angefordert, ruft das Verbindungsobjekt (TConnection) in diesem Fall sofort die Callback-Prozedur der Anwendung auf und übergibt damit die Daten des GPS-Ereignisses an die Anwendung.

Fall 2: Anwendung wird per Nachricht informiert:

Handelt es sich um ein Ereignis, über das die Anwendung benachrichtigt werden will, so wird dieses Ereignis in die dem Verbindungsobjekt zugeordneten Ereigniswarteschlange eingetragen und die Anwendung durch ein *PostMessage()* über das Eintreffen des Ereignisses benachrichtigt. Die Anwendung sollte in der entsprechenden Nachrichten-Behandlungsroutine *GetGpsData()* aufrufen. Damit erhält sie die Daten des GPS-Ereignisses und das Ereignis selbst kann aus der Ereigniswarteschlange entfernt werden.

Die Verwaltung von GPS-Ereignissen in der Ereigniswarteschlange

Ein vom GPS Gerätetreiber dekodiertes Ereignis gelangt über den Funktionsaufruf gpsapi_DeviceResult() zurück an das diesem Gerät zugeordnete TDevice-Objekt. Dieses Objekt verwaltet für jede bestehende Verbindung (TConnection) eine Queue für GPS-Ereignisse. GPS-Ereignisse bleiben solange in dieser Queue bis sie durch den Aufruf von gpsapi_GetGpsData von der Anwendung abgeholt werden. Abb. 22 zeigt einen möglichen Zustand dieser Queue:

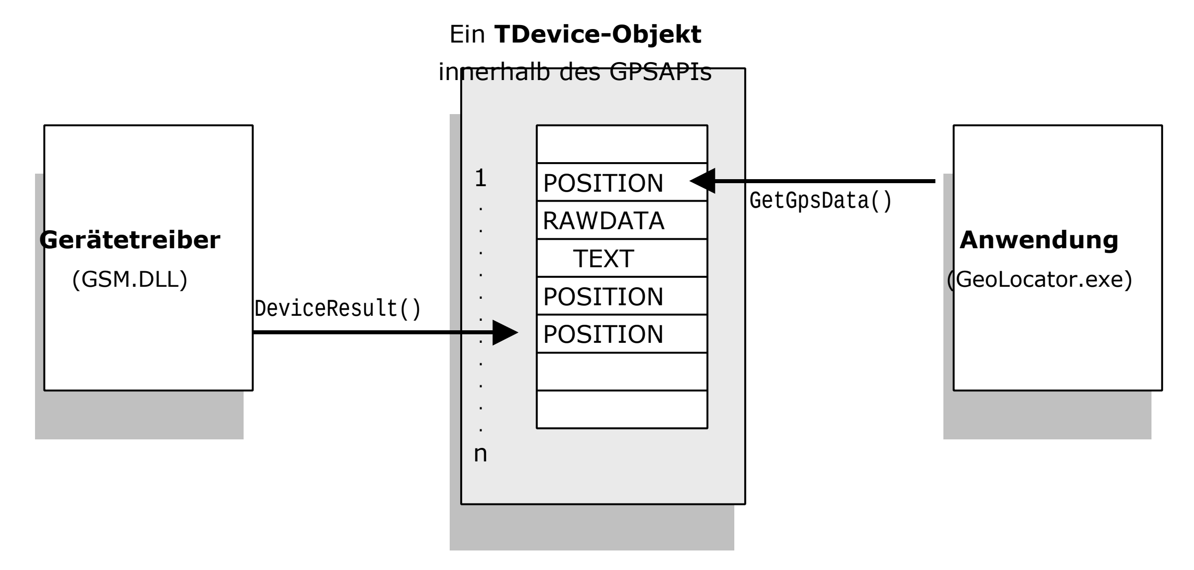


Abb. 22. Zustand der Ereigniswarteschlange zu einem Zeitpunkt t

Jedesmal wenn ein neues Ereignis in der Queue eintrifft, wird die Anwendung mit der entsprechenden API-Nachricht über das Eintreffen und über den Typ des Ereignisses benachrichtigt (bzw. deren Callback-Prozedur aufgerufen). Im Normalfall ruft nun die Anwendung gpsapi_GetGpsData() mit einem Empfangspuffer und dem entsprechenden Ergebnistyp als Parameter auf. Im oben skizzierten Beispiel sollte die Anwendung jetzt folgende Anweisung ausführen:

```
gpsapi_GetGpsData (hConnection,          // connection handle
                   GPSAPIEV_POSITION,    // Ereignis-Typ
                   lpOriginator,         // Puffer für den Absender (optional)
                   lpData,               // Puffer für die GPSAPI_POS Struktur
                   sizeof (GPSAPI_POS)); // Größe des Datenpuffers
```

Es kann an dieser Stelle seitens des GPSAPI‘s nicht ausgeschlossen werden, daß eine Anwendung gpsapi_GetGpsData() nicht, oder mit einem falschen Ereignis-Typ, z.B. GPSAPIEV_TEXT (die Anwendung fordert fälschlicherweise eine Textnachricht an, obwohl sich an erster Stelle der Queue eine Position befindet) aufruft.

Das ist i.a. dann der Fall, wenn eine Anwendung beim Öffnen der Verbindung bestimmte Ereignisse zwar anfordert, aber bei der Benachrichtigung nicht bearbeitet. Damit diese, nicht abgeholten Ereignisse den weiteren Ablauf nicht blockieren oder in der Folge zu einem Datenüberlauf führen, werden solange Ereignisse sequentiell aus der Queue gelöscht, bis ein Ereignis mit dem Ereignis-Typ des Prozeduraufrufes übereinstimmt. Alle gelöschten Ereignisse sind damit verloren.

Es ist daher im Sinne eines guten Programmierstils, wenn beim Öffnen einer Verbindung zum GPS-Device auch nur diejenigen Ereignisse angegeben werden, die auch tatsächlich verarbeitet werden.

Unabängig davon werden in der Ereigniswarteschlange maximal 256 Ereignisse zwischengespeichert. Ab diesem Zeitpunkt wird beim Einlangen eines neuen Ereignisses jeweils das älteste gelöscht.

Datenstrukturen

GPSDEVCAPS

Bevor eine Anwendung eine Verbindung zu einem Empfänger aufbaut, kann es mittels der Funktion *gpsapi_GetDevCaps()* bestimmte Eigenschaften des Empfängers abfragen. Dafür muß die wie folgt definierte Datenstruktur bereitgestellt werden:

```
typedef struct _GPSDEVCAPS {
  char  szName[GPSAPI_MAXDEVNAME];  // the device's name (and manufacturer)
  char  szLibrary[MAX_PATH];        // filename of the device driver
  DWORD dwDriverVersion;            // driver version
  DWORD dwCapabilities;             // capabilities of this device
  int   nPortId;                    // id of serial port (e.g. 1 .. 4)
  DWORD dwBaudrate;                 // 2400 | 9600 | 19200 ...
  DWORD nDevId;                     // unique device id in the current configuration
  union {
    DWORD dwFlags;
    DWORD fLogFile:1;               // log device events?
  };
} GPSDEVCAPS;
```

In *szName* steht nach Aufruf der Funktion der Name (und eventuell auch der Hersteller) des GPS-Empfängers. *szLibrary* beinhaltet Pfad und Name des Gerätetreibers. Diese Angabe ist an und für sich nur für das das GPSAPI selbst von Interesse. Ebenso braucht sich eine Anwendung nicht um *dwDriverVersion*, *nPortId* und *dwBaudrate* kümmern.

Für eine Anwendung sind die Datenfelder *dwCapabilities* und vor allem die *nDevId* wichtig. *dwCapabilities* ist vorgesehen für besondere Eigenschaften des Empfängers (z.B. ob auch Textnachrichten übertragen werden können).

Die *nDevId* wird von der Anwendung unbedingt benötigt, wenn es eine Verbindung zu diesem Empfänger herstellen will.

GPSAPI_POS

Zentrale Datenstruktur des GPSAPI's ist die GPSAPI_POS. Unabhängig vom jeweils verwendeten GPS-Empfänger liefert das GPSAPI Positionen immer in diesem Format:

```
typedef struct {
    long            id;           // the position's id (optional)
    double          gpsTime;      // gps-time in seconds (weekseconds)
    GPSAPI_TIME     gmtTime;      // gmt-time
    GPSAPI_LLHPT    llhPt;        // geogaphic coordinates in WGS84
    GPSAPI_XYZPT    xyzPt;        // transformed xyz-coordinates (z = height)
    double          speed;        // speed in km/h
    double          hdg;          // heading in ø (track)
    double          dop;          // dilution of precision
    unsigned char   dopType;      // 0 = PDOP, 1 = HDOP
    unsigned char   rcvStatus;    // receiver status
    unsigned char   satsTrk;      // number of recently tracked satellites
    unsigned char   satsVis;      // number of recently visible satellites
    unsigned char   quality;      // quality indicator
    char            skId;         // static/kinamatic  ('S' | 'K' | '-')
} GPSAPI_POS;
```

Die meisten Datenelemente der Struktur stammen direkt aus den vom GPS-Empfänger gesendeten Daten. Je nach Empfängertyp können jedoch ein oder mehrere Datenfelder leer (= 0) bleiben.

Das GPSAPI kann bei Bedarf die Positionsdaten, die standardmäßig im Koordinatensystem WGS84 ausgegeben werden, auch sofort in ein frei definierbares, lokales Koordinatensystem transformieren (die transformierten Koordinaten stehen dann im Datenfeld xyzPt).

GPSAPI_SATTRK

Zusätzlich zur Position wird manchmal auch der sog. *Trackingstatus* der einzelnen Satelliten benötigt. Diese Statusinformation enthält im wesntlichen ein Gültigkeitsflag, Signalstärke, und momentane Position des Satelliten. Das GPSAPI definiert dafür die folgende Datenstrukturen:

```
typedef struct {               // tracking status for each channel
  unsigned char satId;         // sat id
  unsigned char mode;          // tracking mode
  unsigned char signal;        // signal strength
  unsigned char chanFlg;       // channnel flag
} GPSAPI_CHN;


/////// tracking status ///////
typedef struct {
  int        chn;              // number of available channels (= 8)
  GPSAPI_CHN channel[8];       // tracking status of each channel
} GPSAPI_SATTRK;
```

In der vorliegenden Implementierung liefert das GPSAPI den Trackingstatus von maximal 8 Satelliten.

GPSAPI_DEVSTATE

Über diese Datenstruktur erhält eine Anwendung Informationen über aktuelle Gerätezustand des GPS-Empfängers und über Erfolg/Mißerfolg ausgeführter Operationen:

```
typedef struct {
  DWORD dwId;                          // device-id
  DWORD dwState;                       // code of current device state
  DWORD dwOperation;                   // latest operation
  DWORD dwResult;                      // result code of last operation
  LONG  lParam;                        // additional parameter (depends on dwOperation)
  char  szText[GPSAPI_MAXSTATETEXT];   // optional description of the messsage
} GPSAPI_DEVSTATE;
```

dwId ist die GeräteId des GPS-Gerätes. *dwState* kann eine der folgenden Werte annehmen:

Wert	Beschreibung
GPSAPI_DS_UNKNOWN	Zustand des Empfänger ist unbekannt
GPSAPI_DS_READY	Gerät ist betriebsbereit
GPSAPI_DS_BUSY	Gerät ist zur Zeit „busy“ und kann keine Daten entgegennehmen.

dwResult gibt Auskunft über Erfolg/Mißerfolg einer gesendeten Positionsanforderung. *dwReference* beinhaltet in diesem Fall dann die von der Anwendung bei Absenden der Anforderung angegebene Id. *szText* kann eine kurze Beschreibung des Gerätezustandes beinhalten.

GPSAPI_LOCATEPARAMS

Diese Datenstruktur bestimmt, in welchen Intervallen ein GPS-Empfänger seine Position an die Anwendung (bzw. den Leitstand) zurücksenden soll. Sie wird von der Funktion *gpsapi_Locate()* verwendet.

```
typedef struct {               // paramters to be provided in 'gpsapi_Locate ()'
  char type;                   // 'p'/'P' for single/cyclic Position, 'r'/'R' for Raw
  long nReports;               // #of automatic reports (device-dependent), 0 if only once
  long intvalSecs;             // interval of reports in seconds (0 for 'continous')
  long daySecsFrom;            // } if!=0, messages should only be sent within this time
  long daySecsTill;            // }       of day (GMT-time expressed in 'seconds of day')
  unsigned char switchStatus;  // optional: switches' status to set (ON/OFF)
  unsigned char switchDelta;   // optional: which switches' status to set
} GPSAPI_LOCATEPARAMS;
```

GPSAPI_ORIGINATOR, GPSAPI_RECIPIENT

Mit dieser Datenstruktur wird ein Empfänger oder Absender einer Position oder einer Textnachricht oder eines anderen Ereignisses definiert:

```
typedef struct {
  DWORD dwId;                          // device-id
  char  szName[GPSAPI_MAXNAME];        // optional
  char  szAddress[GPSAPI_MAXADDRESS];  // (i.e. phone number)

} GPSAPI_ORIGINATOR,
  GPSAPI_RECIPIENT;
```

Schnittstelle

In diesem Abschnitt werden die einzelnen API-Aufrufe erläutert. Man kann grundsätzlich die Funktionen in zwei Gruppen unterteilen:

1. Funktionen, die vom Anwendungsprogramm aufgerufen werden (Herstellen einer Verbindung zu einem Gerät, Festlegen von Geräteoptionen, Einstellung des zu verwendenden Landeskoordinatensystems), Konvertierungsroutinen.
2. Funktionen, die für GPS-Gerätetreiber vorgesehen sind (Übergeben von decodierten GPS-Datensätzen, Ausgabe von Meldungen in die Log-Datei).

1. Die Schnittstelle für Anwendungsprogramme

---

int gpsapi_GPSSetup (HWND hwndOwner);

Die Funktion öffnet den GPS-Setup Dialog. In einer Liste werden alle am Rechner bereits installierten Geräte aufgelistet. Weitere bzw. neue GPS-Geräte können installiert, bestehende konfiguriert oder wieder entfernt werden. Einzustellen ist für jedes Gerät zumindest die serielle Schnittstelle und die Baudrate.

Parameter

HWND hwndOwner
Fenster-Handle des aufrufenden Elternfensters.

Rückgabewert:

IDOK	Der Benutzer hat den „OK“ Schalter betätitgt.
IDCANCEL	Der Benutzer hat den Dialog mit „Abbrechen“ beendet.

Anmerkung

Der Rückgabewert dient lediglich zur Information und hat keine weitere Bedeutung.

---

int gpsapi_GetNumDevs();

Ermittelt die Anzahl der vorhanden, installierten GPS-Geräte.

Rückgabewert:

Anzahl der GPS-Geräte.

int gpsapi_GetDevCaps (int index, GPSDEVCAPS* lpDevCaps);

Ermittelt spezielle Geräteinformation für das angegebene Gerät, u.a. auch eine eindeutige Geräte-ID die für alle weiteren Funktionsaufrufe, die auf GPS-Geräte bezug nehmen, benötigt wird.

Parameter:

int *index*

Index auf ein GPS-Gerät. Gültiger Bereich ist [0..*n*-1], wobei *n* die Anzahl der vorhanden GPS-Geräte ist. *n* kann mit der Funktion **gpsapi_GetNumDevs()** ermittelt werden.

GPSDEVCAPS **lpDevCaps*

Zeiger auf eine GPSDEVCAPS Struktur. Sie ist wie folgt definiert:

```
////////// gps device capabilities //////////
typedef struct _GPSDEVCAPS {
  char  szName[GPSAPI_MAXDEVNAME];  // the device's name (and manufacturer)
  char  szLibrary[MAX_PATH];        // full filename of the device driver
  DWORD dwDriverVersion;            // driver version
  DWORD dwCapabilities;             // capabilities of this device
  int   nPortId;                    // id of serial port (1 | 2 | 3 ...)
  DWORD dwBaudrate;                 // 2400 | 9600 | 19200 ...
  DWORD nDevId;                     // unique device id in the current configuration
  union {
    DWORD dwFlags;
    DWORD fLogFile:1;               // log device events?
  };
} GPSDEVCAPS;
```

Rückgabewert:

TRUE bei Erfolg, FALSE wenn keine Geräteinformationen ermittelt werden konnten (z.B. wenn *index* außerhalb des gültigen Bereiches liegt).

HWND gpsapi_OpenDevStatusWnd (HWND hwndOwner, DWORD nDevId);

Öffnet das Geräte-Statusfenster. In diesem Fenter wird jeweils die letzte eingegangene GPS-Position, Anzahl der sichtbaren und verfolgten Satelliten, DOP-Wert, Richtung, Geschwindig, etc... angezeigt. Die einzelnen Informationen können nach Bedarf ein- bzw. ausgeblendet werden.

Es handelt sich bei diesem Statusfenster um einen nicht modalen Dialog. Er bleibt solange geöffnet, bis der Benutzer ihn explizit schießt.

Parameter:

HWND *hwndOwner*

Ein Fenster-Handle der aufrufenden Anwendung.

DWORD *nDevId*

Gerätde-ID. Ist diese ID gültig, wird das Statusfenster für das entsprechende Gerät geöffnet. Wenn diese ID mit 0 angegeben wird und mehr als ein GPS-Gerät installiert ist, wird der Benutzer aufgefordert, eines der installierten Geräte auszuwählen.

Rückgabewert:

HWND Fenster-Handle des nicht modalen Statusdialoges

---

int gpsapi_OpenDevOptionsDlg (HWND hwndOwner, DWORD nDevId);

Öffnet einen Dialog für spezielle Geräteeinstellungen. Es ist ein Dialog, der vom Gerätetreiber selbst zur Verfügung gestellt wird.

Parameter:

HWND *hwndOwner*

Ein Fenster-Handle der aufrufenden Anwendung.

DWORD *nDevId*

Gerätde-ID des entsprechenden Gerätes. Diese ID kann mit Hilfe der Funktion **gpsapi_GetDevCaps()** ermittelt werden.

Rückgabewert:

IDOK, falls der Dialog mit ‚OK‛ geschlossen wurde, IDCANCEL sonst.

HANDLE gpsapi_Connect (DWORD nDevId,
HWND hwndOwner,
DWORD GpsEventProc,
DWORD dwFlags);

Diese Funktion ermöglicht einer Anwendung eine Verbindung zum angegebenen GPS-Gerät herzustellen. Die aufrufende Anwendung erhält bei erfolgreichem Verbindungsaufbau in der Folge vom GPSAPI Rückmeldungen über verschiedene GPS-Ereignisse (GPS-Positionen, Trackingstatus, eventuell auch Gerätestatusmeldungen oder Textnachrichten).

Diese Rückmeldungen können wahlweise über eine benutzerdefinierte Nachricht (user defined message) oder über eine Callback-Prozedur (GpsEventProc) erfolgen.

Parameter:

DWORD *nDevId*

Gerätde-ID des entsprechenden Gerätes. Diese ID kann mit Hilfe der Funktion **gpsapi_GetDevCaps()** ermittelt werden.

HWND *hwndOwner*

Fenster-Handle, an das alle GPS-Nachrichten über den Windows-Messaging Mechanismus gesendet werden sollen. Falls dieses Handle NULL ist, verwendet das GPSAPI den Callback-Mechanismus, um die aufrufende Anwendung über Ereignisse zu be-

nachrichtigen. In diesem Fall muß der Parameter *GpsEventProc* unbedingt auf eine gültige Callback-Prozedur verweisen.

DWORD *GpsEventProc*

In Abhängigkeit davon, ob der Parameter *hwndOwner* ein gültiges Fensterhandle enthält oder 0 ist, muß *GpsEventProc* entweder eine benutzerdefinierte Nachricht (WM_USERxxx), oder die Adresse einer Callback-Prozedur enthalten.

Verwenden einer benutzerdefinierten Nachricht:

Falls der Benutzer durch Windows-Nachrichten über Ereigsnisse verständigt werden will, erhält er GPS-Ereignisse an das im Parameter *hwndOwner* angegebene Fenster zugesandt. Der WPARAM-Wert enthält dann den Ereignistyp, LPARAM das Handle der Verbindung auf das sich das Ereignis bezieht. Die Daten (z.B. die Positionsdaten) des Ereignisses selbst erhält der Benutzer durch Aufruf von **gpsapi_GetGpsData()**.

Verwenden einer Callback-Funktion:

Falls der Benutzer eine Callback-Funktion verwenden will, muß diese folgendermaßen definiert sein:

```
void (WINAPI *GpsEventProc) (
      GPSAPI_ORIGINATOR* lpOriginator,   // pointer to a ‚originator‘ structure
      UINT uType,                        // type of event
      LPVOID lpData,                     // pointer to buffer containing the event data
      UINT uSize);                       // sizeof referred data
```

In Abhängigkeit des Ereignisses zeigt *lpData* auf eine der folgenden Datenstrukturen:

GPS-Ereignis:	assoziierte Datenstruktur:
GPSAPIEV_POSITION	GPSAPI_POS
GPSAPIEV_SATTRK	GPSAPI_SATTRK
GPSAPIEV_TEXT	GPSAPI_TEXTMSG
GPSAPIEV_DEVSTATE	GPSAPI_DEVSTATE
GPSAPIEV_RAWDATA	GPSAPI_RAWDATA

DWORD *dwFlags*

Dieser Parameter bezeichnet die Ereignistypen, die die Anwendung erhalten möchte. Es kann sich dabei um eine Kombination der folgenden Konstanten handeln:

GPS-Ereignis:	Beschreibung
GPSAPIEV_POSITION	Anwendung wird benachrichtigt, wenn GPS-Positionen eintreffen
GPSAPIEV_SATTRK	Anwendungen wird laufend über den Satelliten-Tracking-Status informiert
GPSAPIEV_TEXT	Eine Textnachricht ist eingetroffen
GPSAPIEV_DEVSTATE	Der Gerätestatus des GPS-Gerätes hat sich geändert
GPSAPIEV_RAWDATA	Ein Roh-Datenformat vom GPS-Empfänger ist eingetroffen

Wenn eine der gewünschten Ereignisse eintrifft, so enthält die an die Hauptanwendung gesandte Nachricht im WPARAM-Wert ebenfalls genau diese Werte.

Rückgabewert:

Im Erfolgsfall liefert die Anwendung eine gültiges Verbdinungs-Handle (connection handle). Dieses Handle wird zum Schließen einer Verbindung **gpsapi_Disconnect()** wieder benötigt. Im Fehlerfall ist der Rückgabewert 0.

BOOL gpsapi_Disconnect (HANDLE hConnection);

Mit dieser Funktion wird die Verbindung zu einem GPS-Empfänger geschlossen.

Parameter:

HANDLE *hConnection*
Verbindungs-Handle, das beim Aufruf von **gpsapi_Connect()** zurückgegeben wird.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

int gpsapi_Locate (DWORD nDevId,
GPSAPI_RECIPIENT* pRecp,
GPSAPI_LOCATEPARAMS* pParams,
DWORD requestId);

Sendet eine Positionsanforderung an das durch pRecp adressierte Fahrzeug.

Parameters:

DWORD *nDevId*
Gerätde-ID des entsprechenden Gerätes.

GPSAPI_RECIPIENT* *pRecp*
Gibt die Adresse des Empfängers (Fahrzeuges) an.

GPSAPI_LOCATEPARAMS* *pParams*
Gibt an, wenn und wie oft das Fahrzeug seine Positionsdaten an den Leitstand senden soll.

DWORD *requestId*
Von der aufrufenden Anwendung angegebener Schlüssel. Diese ID wird vom GPSAPI verwendet um die Anwendung über den Status (gesendet, Fehler, ...) der Positionsanforderung zu benachrichtigen.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

int gpsapi_SendTextMsg (DWORD dwDevId,
GPSAPI_RECIPIENT* pRecp,
LPSTR pszText,
DWORD msgId);

Sendet eine Textnachricht an den angegebenen Empfänger.

Parameter:

DWORD *dwDevId*
Gerätde-ID des entsprechenden Gerätes.

GPSAPI_RECIPIENT* *pRecp*
Gibt die Adresse des Empfängers (Fahrzeuges) an.

LPSTR *pszText*
Text der an das Fahrzeug versandt werden soll. Je nach Gerät kann es unterschiedliche Maximallängen geben (bei SMS kann man maximal 160 Zeichen versenden). Zu lange Zeichenketten werden ohne Rückmeldung abgeschnitten.

DWORD *msgId*
Von der aufrufenden Anwendung angegebener Schlüssel. Diese ID wird vom GPSAPI verwendet um die Anwendung über den Status (gesendet, Fehler, ...) der Nachricht zu benachrichtigen.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

int gpsapi_GetGpsData (HANDLE hConnection, UINT uType,
GPSAPI_ORIGINATOR* lpOriginator,
SYSTEMTIME *sysTime,
LPVOID lpData, UINT uSize);

Mit dieser Funktion liest eine Anwendung das jeweils nächste GPS-Ereignis aus der Ereigniswarteschlange. Normalerweise wird diese Funktion nach Erhalt einer GPS-Ereignisbenachrichtigung aufgerufen, um die GPS-Daten zu entgegenzunehmen.

Anmerkung: Diese Funktion wird nicht benötigt, wenn die Anwendung beim Aufbau der Verbindung eine Callback-Funktion angegeben hat. In diesem Fall werden nämlich die GPS-Daten sofort nach Erhalt der Callback-Funktion übergeben, und nicht in der GPS-Ereigniswarteschlange zwischengespeichert.

Parameter:

HANDLE *hConnection*
Verbindungs-Handle, entspricht dem LPARAM-Wert der GPS-Nachricht.

UINT *uType*
Typ des GPS-Ereignisses. Da die Funktion im Normalfall nur als Antwort auf eine GPS-Nachricht aufgerufen wird, kann der WPARAM-Wert der GPS-Nachricht verwendet werden. Bzgl. Ereignistypen siehe Beschreibung des *lpData*-Parameters.

GPSAPI_ORIGINATOR **lpOriginator*

Zeiger auf den Speicherbereich für den Absender des Ereignisses. Dieser Wert kann NULL sein, wenn der Absender nicht benötigt wird.

SYSTEMTIME **lpSysTime*

Zeiger auf eine Datenstruktur SYSTEMTIME zum Speichern des Zeitpunktes des Eintreffens des Ereignisses. Dieser Wert kann ebenfalls NULL sein, wenn der Zeitpunkt des Eintreffens der Nachricht nicht benötigt wird.

LPVOID *lpData*

Zeiger auf den Speicherbereich für die GPS-Daten. Je nach Ereignistyp kann dies eine der folgenden Datenstrukturen sein:

GPS-Ereignistyp	korrespondierende Datenstruktur
GPSAPIEV_POSITION	GPSAPI_POS
GPSAPIEV_SATTRK	GPSAPI_SATTRK
GPSAPIEV_TEXT	GPSAPI_TEXTMSG
GPSAPIEV_DEVSTATE	GPSAPI_DEVSTATE
GPSAPIEV_RAWDATA	GPSAPI_RAWDATA

UINT *uSize*

Größe des Datenpuffers für die GPS-Daten.

Rückgabewert:

Der Rückgabewert entspricht der Anzahl der sich noch in der Warteschlange befindlichen GPS-Ereignisse. Ist der Rückgabewert kleiner als Null, ist ein Fehler aufgetreten (kein Datensatz in der Queue).

Konvertierungsroutinen

Die folgenden Funktionen bieten Unterstützung für häufig benötigte Konvertierungsaufgaben.

LPSTR gpsapi_Time2Str (GPSAPI_TIME gmtTime, LPSTR lpBuffer);

Gibt die angegebene GPS-Uhrzeit in eine Zeichenkette aus.

Parameter:

GPSAPI_TIME *gmtTime*

Datum und Uhrzeit, die konvertiert werden sollen.

LPSTR *lpBuffer*

Adresse des Puffers, in die die formatierte Zeichenkette geschrieben wird. Dieser Puffer muß mindestens 32 Zeichen lang sein.

Rückgabewert:

Der Rückgabewert ist ein Zeiger auf den Puffer.

LPSTR gpsapi_Lat2Str (double lat, LPSTR lpBuffer);

Konvertiert die gegebene geographische Breite in eine Zeichenkette der Form
N 48° 10‘ 44.1132“

Parameter:

double *lat*
Geographische Breite

LPSTR *lpBuffer*
Adresse des Puffers, in die die formatierte Zeichenkette geschrieben wird. Dieser Puffer muß mindestens 32 Zeichen lang sein.

Rückgabewert:
Der Rückgabewert ist ein Zeiger auf den Puffer.

LPSTR gpsapi_Lon2Str (double lon, LPSTR str);

Konvertiert die gegebene geographische Länge in eine Zeichenkette der Form
E 13° 12‘ 17.8442“

Parameter:

double lon
Geographische Länge

LPSTR *lpBuffer*
Adresse des Puffers, in die die formatierte Zeichenkette geschrieben wird. Dieser Puffer muß mindestens 32 Zeichen lang sein.

Rückgabewert:
Der Rückgabewert ist ein Zeiger auf den Puffer.

Einstellen von Ziel-Koordinatensystemen

Mit den Funktionen zur Koordinatentransformation kann der Anwender angeben, in welches Landeskoordinatensystem die GPS-Positionen umgerechnet werden sollen. Zusätzlich stehen Dialoge zum Definieren von neuen Koordinatensystemen zur Verfügung.

Achtung! Wird innerhalb derselben Anwendung auch das MAPAPI verwendet, so ist zu beachten, daß das MAPAPI dieselbe Datenbasis zur Koordinatentransformation verwendet. Es ist also Vorsicht geboten, wenn Koordinatensysteme verändert oder gar gelöscht werden.

---

int gpsapi_GeoCalculatorDlg (HWND hwndOwner);

---

gpsapi_GeoCalculatorDlg() öffnet einen Dialog mit dem der Benutzer manuell Koordinaten zwischen beliebigen Koordinatensystemen transformieren kann.

Parameter:

HWND *hwndOwner*
Ein Fensterhandle der aufrufenden Anwendung.

Rückgabewert:
(wird nicht verwendet, immer 0)

---

int gpsapi_SelectCoordSystemDlg (HWND hwndOwner);

Öffnet einen Dialog zum Auswählen eines bestimmten Landeskoordinatensystems. In dieses Koordinatensystem werden alle eingehenden GPS-Positionen konvertiert, bevor sie an die Anwendung weitergeleitet werden. Genaugenommen werden die Datenfelder der GPSAPI_XYZPT-Struktur innerhalb der GPSAPI_POS-Struktur gesetzt. Die ursprünglichen WGS84-Koordinaten bleiben in der GPSAPI_LLHPT-Struktur (Länge/Breite/Höhe) auch noch erhalten.

Parameter:

HWND *hwndOwner*
Ein Fensterhandle der aufrufenden Anwendung.

Rückgabewert:
(wird nicht verwendet, immer 0)

int gpsapi_DefineCoordSystemDlg (HWND hwndOwner);

Öffnet einen Dialog, mit dem ein neues Landeskoordinatensystem definiert, oder ein nicht mehr verwendetes gelöscht werden kann.

Parameter:

HWND *hwndOwner*
Ein Fensterhandle der aufrufenden Anwendung.

Rückgabewert:
(wird nicht verwendet, immer 0)

int gpsapi_DefineDatumDlg (HWND hwndOwner);

Mit diesem Dialog kann eine neues Datum festgelegt und bestehende bearbeitet oder gelöscht werden.

Parameter:

HWND *hwndOwner*
Ein Fensterhandle der aufrufenden Anwendung.

Rückgabewert:
(wird nicht verwendet, immer 0)

int gpsapi_DefineEllipsoidDlg (HWND hwndOwner);

Mit diesem Dialog kann eine neues Ellipsoid definiert werden.

Parameter:

HWND *hwndOwner*
Ein Fensterhandle der aufrufenden Anwendung.

Rückgabewert:
(wird nicht verwendet, immer 0)

2. Die Schnittstelle für GPS-Gerätetreiber

Die nun folgenden Funktionen werden üblicherweise nur von GPS-Gerätetreibern aufgerufen, um dem GPSAPI bestimmte Ereignisse oder Ergebnisse zu übergeben, oder um Daten über die serielle Schnittstelle an das Gerät zu senden.

---

int gpsapi_DeviceResult (HANDLE hDevice,
UINT uType,
GPSAPI_ORIGINATOR* lpOriginator,
SYSTEMTIME *lpSysTime,
LPVOID lpData,
UINT uSize);

Die gpsapi_DeviceResult Funktion wird von einer GPS-Gerätetreiber DLL aufgerufen, sobald im Datenstrom ein gültiger GPS-Datensatz gefunden wurde.

Parameter:

HANDLE *hDevice*
Handle des GPS-Gerätes, von dem der Datensatz gelesen wurde.

UINT *hType*
Art des Ereignisses; eine Konstante vom Typ `GPSAPIEV_xxx`.

GPSAPI_ORIGINATOR **lpOriginator*
Zeiger auf Datenstruktur des Absender des Ereignisses an (bei einer GPS-Position würde im Adreßfeld z.B. die Mobilnummer des Fahrzeuges stehen).

SYSTEMTIME **lpSysTime*
Zeitpunkt des Ereignisses

LPSTR *lpData*
Zeiger auf den entsprechenden Ereignis-Datensatz (eine Struktur vom Typ GPSAPI_xxx)

UINT *uSize*
Größe des Datensatzes in Bytes, auf den *lpData* verweist.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

---

void _export WINAPI gpsapi_SetTimeOut (HANDLE hDevice,
DWORD dwTimeOut);

Diese Funktion setzt den aktuellen Timeout-Wert für ein bestimmtes Gerät. Antwortet das Gerät auf einen Befehl nicht innerhalb der angegebenen Zeitspanne, erhält die Anwendung eine Timeout-Fehlermeldung.

Parameter:

HANDLE *hDevice*
Handle des GPS-Gerätes

DWORD *dwTimeOut*
Zeitspanne in 100 Millisekunden (10 = 1 Sekunde). Wird der Wert auf 0 gesetzt, wird der Timeout-Mechanismus deaktiviert.

Rückgabewert:
keiner.

BOOL gpsapi_Write (HANDLE hDevice,
LPSTR lpBuffer,
DWORD dwBytesToWrite,
LPDWORD lpBytesWritten);

Mit dieser Funktion kann der Gerätetreiber Daten an das Gerät senden.

Parameter:

HANDLE *hDevice*
Handle des GPS-Gerätes.

LPCVOID *lpBuffer*
Adresse der Daten, die gesendet werden sollen.

DWORD *dwBytesToWrite*
Größe des Datenpuffers in Bytes.

LPDWORD *lpBytesWritten*
Adresse für Anzahl der Zeichen die gesendet wurden.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

BOOL gpsapi_WriteLog (HANDLE hDevice, LPSTR lpBuffer);

Mit gpsapi_WriteLog kann die Gerätetreiber DLL beliebige Meldungen oder Warnungen aufzeichnen. Eine Prokollierung ist allerdings nur dann möglich, wenn bei den Geräteoptionen dieses Gerätes die Option „Prokolldatei aufzeichnen" aktiviert wurde.

Diese Funktion schreibt den angegebnen String mit Uhrzeit und Datum versehen in das dem GPS-Gerät zugeordnete Logfile. Das Logfile hat den gleichen Namen wie der Gerätetreiber des GPS-Gerätes, allerdings mit der Dateiendung „.log". Logfiles werden immer in das temporäre Verzeichnis des lokalen Rechners geschrieben.

Parameter:

HANDLE *hDevice*
Handle des GPS-Gerätes.

LPSTR *lpBuffer*
Adresse des zu protokolliernden Textes.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

HANDLE gpsapi_GetDevice (DWORD nDevId);

Die Funktion **gpsapi_getDevice()** ermittelt anhand der Device-ID das aktuelle Handle des Gerätes.

Parameter:

DWORD *nDevId*
Eindeutige Geräte-ID

Rückgabewert:
Die Funktion liefert das Handle des Gerätes. Der Rückgabewert ist NULL, falls das Gerät nicht gefunden wurde.

DWORD gpsapi_GetDeviceId (HANDLE hDevice);

Diese Funktioniert liefert zu einem gegebenen Gerätehandle die Geräte-ID.

Parameter:

DWORD nDevId
Eindeutige Geräte-ID

Rückgabewert:
Die Funktion liefert das Geräte-ID des Gerätes. Der Rückgabewert ist 0, falls das Gerät nicht existiert.

BOOL gpsapi_SetDevValue (DWORD nDevId, LPSTR lpszValue, DWORD dwType, void *pData, DWORD cbData);

Diese Funktion schreibt den angegebenen Wert des GPS Gerätes an die entsprechende Stelle der Registry: Einstellungen für GPS-Geräte werden im Zweig `HKEY_LOCAL_MACHINE\SOFTWARE\Communication & Navigation\GPSAPI\GPSDevices` abgelegt. Für jedes Gerät wird ein eigener Schlüssel angelegt, der der Geräte-ID entspricht (z.B. 00000001 für

das erste, 00000002 für das zweite Gerät, das an den lokalen Computer angeschlossen ist).

Parameters:

DWORD nDevId
Geräte-Id.

LPSTR pszValue
Adresse des Wertes, der gesetzt werden soll

DWORD dwType
Gibt den Datentyp an, der gespeichert werden soll. Die möglichen Werte sind ident mit denen, die beim entsprechenden Parameter der Windows-API-Funktion *RegSet-ValueEx()* verwendet werden können.

void* pData
Zeiger auf den Puffer, der die zu speichernden Daten enthält.

DWORD cbData
Größe des Datenpuffers in Bytes.

Rückgabewert:
TRUE bei Erfolg, FALSE sonst.

BOOL gpsapi_GetDevValue (DWORD nDevId, LPSTR lpszValue, void *pData, DWORD cbData);

Liest einen Wert des angegebenen GPS-Gerätes aus der Registry.

Parameters:

DWORD nDevId
Geräte-Id.

LPSTR pszValue
Adresse des Wertes der gelesen werden soll

void* pData
Zeiger auf den Puffer, der die zu lesenden Daten aufnehmen soll.

DWORD cbData
Größe des Datenpuffers in Bytes.

Rückgabewert:
TRUE bei Erfolg, FALSE sonst.

Dateireferenz

Die unten aufgelisteten Dateien werden zum Compilieren der GPSAPI.DLL benötigt:

Datei	**Beschreibung**
Verzeichnis *Source:*	
GpsAPI.h	Definition aller exportieren DLL-Funktionen
GpsTypes.h	Definition aller öffentlich bekannten Datenstrukturen
Private.h	Definition aller globalen nicht-exportierten Funktionen und Datenstrukturen
GpsAPI.cpp	Implementierung der exportierten und nicht-exportieren Funktionen
Device.h/.cpp	TDevice, generisches GPS-Empfänger Objekt
ComPort.h/.cpp	TComPort, Objekt zum Steuern von seriellen Schnittstellen unter Win32
Connect.h/.cpp	TConnection, Objekt, das die Zuordnung von Gps-Geräten (TDevice-Objekten) zu „Verbrauchern“ sicherstellt
DGpsSetp.h/.cpp	Allgemeiner Setupdialog für GPS-Geräte
DGpsStat.h/.cpp	Statusanzeige für GPS-Empfänger
DSelDev.h/.cpp	Hilfsdialog zum Auswählen installierter GPS-Geräte
Shared.h/.cpp	Shared Memory Object; Speicherbereich, auf den alle Anwendungen, die die DLL verwenden, Zugriff haben
gpsapi.def	Projekt-Definitionsdatei
Verzeichnis *Res:*	
GpsAPI.rh	Definitionen und Konstanten für alle Resourcen
gapi_gr.rc	Beinhaltet alle sprachabhängigen (deutschen) Resourcen (Dialoge, Strings, ...)
Bitmaps.rc	Bitmaps, Icons, Cursors
Verzeichnis *misc/Source:*	
GeoDbase.h/.cpp	TGeoDatabase, Objekt zur Kapselung der DLL-Aufrufe zur Koordinatentransformation
log.h/.cpp	Logfile-Funktionen (für Debugging Zwecke)
LoadDLL.h/.cpp	eigenes TDll-Objekt, kapselt LoadLibrary, FreeLibrary
RegEx.h/.cpp	TRegistry-Objekt, vereinfacht Zugriff auf Registry
coordcvt.h/.cpp	Koordinatenkonvertierungsfunktionen
gcalcex.h/.cpp	Erweiterte Funktionen für den Zugriff auf BlueMarbles Koordinatensystem-Datei (z.B. Löschen von Einträgen)
syseldlg.h/.cpp	Dialog zum Auswählen eines Koordinatensystems
cosysdlg.h/.cpp	Dialog zum Bearbeiten/Definieren von Koordinatensystemen
datumdlg.h/.cpp	Dialog zum Bearbeiten/Definieren eines geodätischen Datums
ellipdlg.h/.cpp	Dialog zum Bearbeiten/Definieren von Ellipsoiden
geocalc.h/.cpp	„GeoCalculator“, Koordinaten-Transformationsdialog
geodg_gr.rh/.rc	Resourcen der Koordinatendialoge

GPS Gerätetreiber

Ein GPS-Gerätetreiber enthält im wesentlichen nur einen Konfigurationsdialog, um empfängerspezifische Einstellungen vornehmen zu können, sowie eine Decodierroutine, an die alle von der Schnittstelle des GPS-Empfängers eingelesenen Daten weitergeleitet werden. Aufgabe der Decodierungsfunktion ist es den Datenstrom nach gültigen GPS-Datensätzen zu durchsuchen und erkannte Datensätze an das GPSAPI zurückzuliefern.

Wird z.B. eine GPS-Position erkannt, füllt der GPS-Treiber die Datenstruktur GPSAPI_POS mit entsprechenden Werten und gibt sie an die GPSAPI.DLL zurück, die sich darum kümmert, diese Position an alle offenen Verbindungen weiterzuleiten.

Der Gerätetreiber selbst braucht sich also weder um die Handhabung der seriellen Schnittstelle, noch um Details der Rückgabe der gewonnen Daten kümmern.

Für jeden verwendeten Gerätetyp wird eine eigene DLL, mit der Endung .GPS erstellt. Dieses Dateiendung wird vom GPSAPI dazu verwendet, um nach verfügbaren Gerätetreibern zu suchen. Ein Hinzufügen von neuen Gerätetreibern gestaltet sich daher relativ einfach: Man muß nur die neue Geräte-DLL in dasselbe Verzeichnis wie das GPSAPI kopieren. Dann können bereits Geräte dieses Types installiert werden.

Im Umfang des vorliegenden Gesamtsystems sind drei Gerätetreiber enthalten:

- **NMEA.GPS**
  Ein Gerätetreiber für Empfänger, die die Daten im NMEA-Format ausgeben.
- **ONCORE.GPS**
  Für Empfänger der Motorola Oncore Serie.
- **TARGET.GPS**
  Zur Ortung von Empfängern über ein an den PC angeschlosses GSM-Gerät. Auf der Gegenseite (im Fahrzeug) muß ebenfalls ein GSM-Modul und ein GeoTracker der Firma Communication & Navigation angeschlossen sein.

Schnittstelle der Gerätetreiber

Auf die Besonderheiten der einzelnen Gerätetreiber wird an geeigneter Stelle noch genauer eingegangen werden. Die allen Gerätetreibern gemeinsame Schnittstelle wird nun in diesem Abschnitt beschrieben. Sie ist in der Datei GPSDEV.H definiert und muß in allen Gerätetreibern implementiert sein.

---

int gpsdev_GetDevCaps (GPSDEVCAPS *lpGpsDevCaps);

Ermittelt spezielle Eigenschaften des angeschlossenen GPS-Gerätes. Die Funktion wird vom GPSAPI nur einmal, und zwar beim Laden der DLL aufgerufen.

Parameter:

GPSDEVCAPS *lpGpsDevCaps

Zeiger auf eine GPSDEVCAPS Datenstruktur:

```
typedef struct _GPSDEVCAPS {
  char  szName[GPSAPI_MAXDEVNAME];  // the device's name (and manufacturer)
  char  szLibrary[MAX_PATH];        // filename of the device driver
  DWORD dwDriverVersion;            // driver version
  DWORD dwCapabilities;             // capabilities of this device
  int   nPortId;                    // id of serial port (e.g. 1 .. 4)
  DWORD dwBaudrate;                 // 2400 | 9600 | 19200 ...
  DWORD nDevId;                     // unique device id in the current configuration
  union {
    DWORD dwFlags;
    DWORD fLogFile:1;               // log device events?
  };
} GPSDEVCAPS;
```

Zu beachten ist, daß vom Gerätetreiber nur die ersten vier Datenfelder initialisiert werden müssen. Die anderen Datenfelder können gesetzt werden – sie werden vom GPSAPI dann als Standardwerte verwendet.

Rückgabewert:

TRUE bei Erfolg, FALSE sonst.

---

int gpsdev_OptionsDlg (HWND hwndOwner, DWORD dwDeviceId);

Öffnet einen Dialog für spezielle, geräteabhängige Einstellungen. Je nach verwendetem Gerät kann es sich um ganz unterschiedliche Dialoge handeln. Die Einstellungen werden mittels der Funktion `gpsapi_SetDevValue()` im selben Zweig der Registry abgelegt, wie die allgemeinen Einstellungen des Empfängers. Es handelt sich dabei um den Zweig:

`HKEY_LOCAL_MACHINE\SOFTWARE\Communication & Navigation\GPSAPI\GPSDevices\00000001`

„0000001" ist dabei Identifikationsnummer des GPS-Gerätes.

Parameter:

HWND *hwndOwner*
Fensterhandle der aufrufenden Anwendung

DWORD *dwDeviceId*
Geräte-ID des zu konfigurierenden Gerätes. Wird benötigt, um dem GPSAPI mitzuteilen, in welchem Zweig der Registry die Einstellungen abzulegen sind.

Rückgabewert:
(nicht verwendet, immer 0)

int gpsdev_Initialize (HANDLE hDevice, LPSTR lpWorkSpace);

Diese Funktion wird vom GPSAPI aufgerufen, unmittelbar nachdem die serielle Schnittstelle, an die das GPS-Gerät angeschlossen ist, geöffnet wurde. Sie soll vom Treiber dazu verwendet werden, um die Initialisierung des Gerätes einzuleiten.

Parameter:

HANDLE *hDevice*
Gerätehandle des geöffneten Gerätes.

LPSTR *lpWorkSpace*
Zeiger auf einen privaten, dem Gerät zugeordneten Arbeitsspeicher. Dieser Puffer wird vom GPSAPI für jeden GPS-Empfänger reserviert und ist standardmäßig 1024 Byte groß. Daten, die vom Gerätetreiber in diesen Puffer geschrieben werden, bleiben von Funktionsaufruf zu Funktionsaufruf erhalten. Er kann also vom Treiber dazu verwendet werden, um bestimmte Einstellungen oder Zwischenergebnisse zu speichern.

Rückgabewert:
TRUE bei Erfolg, FALSE sonst.

int gpsdev_TimeOut (HANDLE hDevice, LPSTR lpWorkSpace);

Diese Funktion wird vom GpsAPI aufgerufen, wenn der TimeOut-Wert des zuletzt gesendeten Befehles überschritten wurde. Der Gerätetreiber kann daraufhin eine entsprechende Fehlerbehandlung einleiten, oder eine entsprechende Fehlermeldung an die Anwendung senden.

Parameters:

HANDLE *hDevice*
Handle des betroffenen Gerätes.

LPSTR *lpWorkSpace*
Zeiger auf den privaten Arbeitsspeicher.

Rückgabewert:
immer 0 (wird in der aktuellen Version nicht verwendet).

int gpsdev_Decode (HANDLE hDevice,
LPSTR lpBuffer,
DWORD dwSize,
LPDWORD lpParsedBytes,
LPSTR lpWorkSpace);

Die Funktion gpsdev_Decode wird vom GPSAPI aufgerufen, sobald Daten an der seriellen Schnittstelle eingetroffen sind.

Parameter:

HANDLE *hDevice*
Handle des aktuellen Geräteobjektes.

LPSTR *lpBuffer*
Zeigt immer auf das erste noch nicht decodierte Zeichen im dem Gerät zugordneten Empfangspuffer.

DWORD *dwSize*
Anzahl der Zeichen im Empfangspuffer.

LPDWORD *lpParsedBytes*
Adresse der Variable für die Anzahl der vom Treiber verarbeiteten Zeichen. Dieser Wert muß vom Gerätetreiber gesetzt werden, damit der Speicherbereich vom GPSAPI wieder für neue Daten verwendet werden kann.

LPSTR *lpWorkSpace*
Zeiger auf den privaten Arbeitsbereich des Treibers. Hier können Zwischenergebnisse oder bestimmte Gerätezustände zwischengespeichert werden. Sie bleiben bis zum nächsten Funktionsaufruf erhalten.

Rückgabewert:
(nicht verwendet, immer 0)

int gpsdev_Locate (HANDLE hDevice,
DWORD msgId,
GPSAPI_RECIPIENT *lpRecipient,
GPSAPI_LOCATEPARAMS *lpLocateParams,
LPSTR lpWorkSpace);

Mit dieser Funktion sendet der Gerätetreiber eine Positionsanforderung an das durch *lpRecipient* bezeichnete Fahrzeug.

Falls es sich beim Gerät um einen lokal angeschlossenen GPS-Empfänger handelt, muß diese Funktion nicht implementiert sein, da GPS-Empfänger üblicherweise GPS-Positionen in einem bestimmten Takt ausgeben. Die Taktrate wäre in diesem Fall im Dialog **gpsdev_OptionsDlg()** einzustellen.

Parameter:

HANDLE *hDevice*
Handle des aktuellen Geräteobjektes.

DWORD *msgId*

Eine von der aufrufenden Anwendung definierte ID. Diese ID muß bei Statusmeldungen (Erfolg/Mißerfolg der Sendevorganges) vom Gerätetreiber zurückgegeben werden, damit die Anwendung die Positionsanforderung eindeutig identifizieren kann.

GPSAPI_RECIPIENT **lpRecipient*

Spezifiziert die Adresse des Empfängers (Fahrzeuges).

GPSAPI_LOCATEPARAMS **lpLocateParams*

Zeiger auf die Datenstruktur, die angibt wann, wie oft und in welchen Intervallen das Fahrzeug seine Positionsdaten an den Leitstand zurücksenden soll.

LPSTR *lpWorkSpace*

Zeiger auf den privaten Arbeitsbereich des Treibers.

Rückgabewert:

TRUE bei Erfolg, FALSE sonst.

```
int gpsdev_SendTextMsg  (HANDLE hDevice,
                         DWORD msgId,
                         GPSAPI_RECIPIENT*,
                         LPSTR pszText,
                         LPSTR lpWorkSpace);
```

Diese Funktion sendet eine Textnachricht an einen durch *pRecipient* bezeichneten Empfänger. Diese Funktion ist nur implementiert, sofern das angeschlossene Gerät (z.B. ein GSM-Handy) überhaupt in der Lage ist Textnachrichten zu versenden.

Parameter:

HANDLE *hDevice*

Handle des aktuellen Geräteobjektes.

DWORD *msgId*

Eine von der aufrufenden Anwendung definierte ID. Diese ID muß bei Statusmeldungen (Erfolg/Mißerfolg der Sendevorganges) vom Gerätetreiber zurückgegeben werden, damit die Anwendung die Textnachricht eindeutig identifizieren kann.

GPSAPI_RECIPIENT **lpRecipient*

Spezifiziert die Adresse des Empfängers (Fahrzeuges).

LPSTR *pszText*

Zeiger auf die zu sendende Textnachricht. Ggf. muß dieser Text (im ANSI Zeichensatz) vom Treiber in den vom Gerät verwendeten Zeichensatz umgewandelt werden.

LPSTR *lpWorkSpace*

Zeiger auf den privaten Arbeitsbereich des Treibers.

Rückgabewert:

TRUE bei Erfolg, FALSE sonst.

Spezielle Gerätetreiber

In diesem Abschnitt wird die Implementierung der drei bestehenden Gerätetreiber dokumentiert. Breiteren Raum wird der *Target.gps*-Treiber einnehmen, weil er der zur Fernortung wichtigste Gerätetyp ist.

NMEA.GPS

Dieser Gerätetreiber ist in der Lage Daten von allen Empfänger zu decodieren, die die Daten gemäß der Spezifikation der National Marine Electronics Association (NMEA) liefern. Inzwischen gibt es mehrere NMEA-Spezifikationen. Der folgenden Abschnitt beschränkt sich auf die NMEA 0183-Spezifikation, die von vielen GPS-Empfängern unterstützt wird.

Gemäß NMEA 0183 werden die Daten über eine serielle Schnittstelle mit 4.800 Baud, acht Datenbits, einem Stopbit und ohne Parität gesendet. Jeder Datensatz beginnt mit einem Dollar Zeichen. Darauf folgt die Satzkennung. Sie kennzeichnet den Inhalt der nachfolgenden Daten, die jeweils durch Beistriche getrennt sind. Nach dem letzten Datum folgt ein Stern als Endezeichen und eine Kontrollsumme. Die Kontrollsumme ist der XOR-Wert aller Zeichen, beginnend mit dem 1. Zeichen nach dem Dollar und endend mit dem letzten Zeichen vor dem Stern. Abgeschlossen wird der Datensatz mit *<CR><LF>* (hex 0D und hex 0A). Die maximale Länge eines Datensatzes ist mit 79 Zeichen begrenzt.

In der vorliegenden Implementierung wertet der NMEA.GPS Treiber die $GPGGA- und die $GPRMC-Datensätze aus. Sie sind wie folgt definiert:

$GPGGA,hhmmss.ss,ddmm.mmm,n,dddmm.mmm,e,q,ss,y.y,a.a,z,g.g,z,t.t,iiii*CC<CR><LF>

Der $GPGGA-Datensatz liefert u.a. Zeitpunkt der Positionsbestimmung (UTC), geographische Länge und Breite, Anzahl der verwendeten Satelliten, HDOP und einen Qualitätsindikator.

Parameter:
(Auszug aus der NMEA 0183 Spezifikation)

hhmmss.ss – UTC of position fix

hh – hours	(00 .. 24)
mm – minutes	(00 .. 59)
ss.ss – seconds	(00.00 .. 59.99)

ddmm.mmm,n – laittude

dd – degrees	(00 .. 90)
mm.mmm – minutes	(00.000 .. 59.999)
n – direction	N – north
	S – south

dddmm.mmm,e – longitude

ddd – degrees	(000 .. 180)
mm.mmm – minutes	(00.000 .. 59.999)
e – direction	E – east
	W – west

q – GPS quality indicator	0 – GPS not available 1 – GPS available 2 – GPS differential fix
ss – Number of satellites being used	(0 .. 12)
y.y – HDOP	
a.a,z – antenna height	
a.a – height	
z – units	M – meters
g.g,z – geoidal separation	
g.g – height	
z – units	M – meters
t.t – age of differential data	
iii – differential reference station ID (0000 .. 1023)	
CC – checksum	(hex 00 .. 7F)

$GPRMC,hhmmss.ss,a,ddmm.mmm,n,dddmm.mmm,w,z.z,y.y,ddmmyy,d.d,v*CC<CR><LF>

Der $GPRMC-Datensatz liefert zusätzlich zur Position auch die momentane Geschwindigkeit und die Richtung. Daher wird er zusätzlich zum $GPGGA-Datensatz ausgewertet. Die Geschwindigkeit wird in Knoten (knots) ausgegeben. Der Umrechnungsfaktor in km/h beträgt 1.852 (1 knoten = 1.852 km/h).

Parameter:
(Auszug aus der NMEA 0183 Spezifikation)

hhmmss.ss – UTC time of position fix	
hh – hours	(00 .. 24)
mm – minutes	(00 .. 59)
ss.ss – seconds	(00.00 .. 59.99)
a – status	A – valid V – invalid
ddmm.mmm,n – laittude	
dd – degrees	(00 .. 90)
mm.mmm – minutes	(00.000 .. 59.999)
n – direction	N – north S – south
dddmm.mmm,e – longitude	
ddd – degrees	(000 .. 180)
mm.mmm – minutes	(00.000 .. 59.999)
e – direction	E – east W – west

`z.z` – speed over ground (knots)	
`y.y` – track made good (reference to true north)	(0.0 .. 359.9)
`ddmmyy` – UTC date of position fix	
`dd` – day	(01 .. 31)
`mm` – month	(01 .. 12)
`yy` – year	(00 .. 99)
`d.d` – magnetic variation (degrees)	(0.0 .. 180.0)
`v` – variation sense	E – east W – west
`CC` – checksum	(hex 00 .. 7F)

Anmerkung:

Die GPSAPI_POS-Datenstruktur wird immer erst nach dem Eintreffen beider Datensätze an das Hauptmodul zurückgegeben, da zum Setzen aller Datenfelder beide benötigt werden.

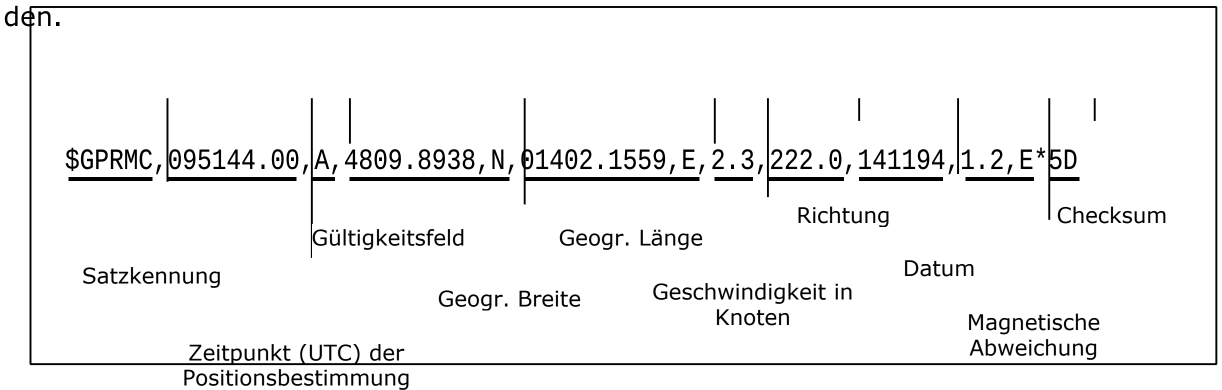


Abb. 23. Beispiel für einen $GPRMC-Datensatz

Dateireferenz NMEA.GPS

Folgende Dateien werden zum Compilieren der NMEA-DLL benötigt:

Datei	**Beschreibung**
Verzeichnis *Source:*	
GpsDev.h	Headerdatei für alle GPS-Gerätetreiber
NMEA.cpp	Implementierung der NMEA-spezifischen Funktionen
NMEA.def	Projektdefinitionsdatei
Verzeichnis *Res:*	
nmea_gr.rh/.rc	Resource-Dateien (deutsch)
Verzeichnis *misc/Source:*	
log.h/.cpp	Logfile-Funktionen (für Debugging Zwecke)
GpsAPI.lib	Bibliothek zum Linken des GpsAPI's (Loadtime-Linking)

ONCORE.GPS

Der ONCORE.GPS Treiber wurde für die Motorola Oncore GPS-Empfänger-Serie entwickelt. Die Daten werden über eine serielle Schnittstelle je nach Gerät entweder mit 9.600 oder 19.200 Baud, acht Datenbits, einem Stopbit und ohne Parität gesendet. Jeder Datensatz beginnt mit zwei at-Symbolen („@@“) und einer Satzkennung, die ebenfalls aus zwei Zeichen besteht. Die Satzkennung bestimmt Länge und Inhalt des nachfolgenden binären Datenstroms. Abgeschlossen wird jeder Datensatz mit einer Prüfsumme (mit der Länge von einem Byte) und *<CR><LF>* (hex 0D und hex 0A).

Es wird an dieser Stelle nur der wichtigste Datensatz (der @@Ba-Datensatz für 6-Kanal-, bzw. der @@Ea-Datensatz für 8-Kanal-Empfänger) angeführt. Aus diesen Datensätzen werden die Daten für die GPSAPI_POS-Datenstruktur entnommen. Bezüglich aller anderen Datensätze sei auf das Handbuch des Empfängers hingewiesen (*Oncore User's Guide* von Motorola).

@@BamdyyhmsffffaaaaoooohhhhmmmmvvhhddtntimsdimsdimsdimsdimsdimsdsC<CR><LF>

Jeder Buchstabe repräsentiert jeweils ein Byte. Länge der Nachricht: 68 Bytes.

Parameter:
(Auszug aus dem „Oncore User's Guide“ von Motorola)

Date

m – month	1 .. 12
d – day	1 .. 31
yy – year	1900 .. 2079

Time

h – hours	0 .. 23
m – minutes	0 .. 59
s – seconds	0 .. 60
ffff – fractional seconds	0 .. 999.999.999 (= 0,0 to 0,999999999)

Position

aaaa – latitude in msec	-324.000.000 .. +324.000.000 (-90° to +90°)
oooo – longitude in msec	-648.000.000 .. +648.000.000 (-180° to +180°)
hhhh – height in cm (GPS, ref ellipsoid)	-100.000 .. +1.800.000 (-1.000,00 to +18.000,00 meters)
mmmm – height in cm (MSL ref)	-100.000 .. +1.800.000 (-1.000,00 to +18.000,00 meters)

Velocity

vv – velocity in cm/sec	0 .. 51.4000 (0 to 514,0 m/sec)
hh – heading (true north. res 0.1°)	(0,0 to 359,9 deg)

Geometry

dd – current DOP (0,1 res)	0 .. 999 (0,0 to 99,9 DOP) (0 – not computable, or position-hold, or position prop)
t –DOP type	0 – PDOP (in 3D mode) 1 – HDOP (in 2D mode)

Satellite visibility and tracking status

n – number of visible satellites 0 .. 12
t – number of tracked satellites 0 .. 6

For each of six receiver channels:

i – sat ID 0 .. 37
m – channel tracking mode 0 .. 8

- 0 – Code Search
- 1 – Code Acquire
- 2 – AGC Set
- 3 – Freq Acquire
- 4 – Bit Sync Detect
- 5 – Message Sync Detect
- 6 – Satellite Time Avail
- 7 – Ephemeris Acquire
- 8 – Avail for Position

s – Signal Strength 0 .. 255
d – Channel Status Flag, each bit represents on of the following:

(msb) Bit 7: Using for Position Fix
Bit 6: Satellite Momentum Alert Flag Set
Bit 5: Satellite Anti-Spoof Flag Set
Bit 4: Satellite Reported Unhealthy
Bit 3: Satellite Reported Inaccurate (> 16 meters)
Bit 2: (spare)
Bit 1: (spare)
(lsb) Bit 0: Parity Error

Receiver Status Message

s – receiver status:

(msb) Bit 7: Position Propagate mode
Bit 6: Poor Geometry (DOP > 20)
Bit 5: 3D fix
Bit 4: Altitude Hold (2D fix)
Bit 3: Acquiring Satellites/Position Hold
Bit 2: Differential
Bit 1: Insufficient visible satellites (< 3)
(lsb) Bit 0: Bad Almanac

C – *Checksum*

Der @@Ea-Datensatz ist wie der @@Ba-Datensatz definiert, allerdings enthält er die Satellitendaten für 8 Satelliten, und ist daher um zwei Satellitendatenfelder (imsd) länger.

Dateireferenz ONCORE.GPS

Folgende Dateien sind dem Oncore-Gerätreiber zugeordnet:

Datei	Beschreibung
Verzeichnis *Source:*	
GpsDev.h	Headerdatei für alle GPS-Gerätetreiber
Oncore.h/.cpp	Implementierung der Oncore-spezifischen Funktionen
DOncOpt.h/.cpp	Oncore-Setup Dialog
fill.h/.cpp	enthält eine Hilfsfunktion zum Füllen der GPSAPI_POS-Struktur
Oncore.def	Projektdefinitionsdatei
Verzeichnis *Res:*	
oncor_gr.rh/.rc	Resource-Dateien (deutsch)
oncr_bmp.rc	Icons und Bitmaps
Verzeichnis *misc/Source:*	
log.h/.cpp	Logfile-Funktionen (für Debugging Zwecke)
Verzeichnis *.\exe:*	
GpsAPI.lib	Bibliothek zum Linken des GpsAPI's (Loadtime-Linking)

TARGET.GPS – Ortung via GSM

Beim sog. „Target"-Gerät handelt es sich um ein GPS-Gerät, das eigentlich kein GPS-Gerät ist. Tatsächlich läßt sich mit dem Treiber jedes beliebige GSM-Gerät ansprechen, sofern es dem ETSI Standard GSM 07.07 (für AT-Kommandos) entspricht.

Sinnvoll einsetzen läßt sich dieser Treiber allerdings nur im Zusammenspiel mit dem *GeoTracker* von C&N. Dieser Empfänger ist in der Lage die gesendeten Positionsanforderungen zu bearbeiten und entsprechend zu beantworten.

Der Treiber wurde entwickelt und getestet mit dem GSM Modul *Falcom A1* der Firma Funkanlagen Leipoldt OHG. Dieses Gerät besitzt eine RS232 Schnittstelle und ist daher ohne weiteres an jeden PC anschließbar.



Abb. 24. Das GSM-Modul ‚Falcom A1' (Foto: Funklanlagen Leipoldt OHG)

Die Beschreibung des TARGET.GPS Gerätetreibers gliedert sich in folgende Unterkapitel:

- **Datenübertragung**
  Anschluß des GSM-Moduls an den PC, allg. Hinweise über Rückmeldungen des GSM-Moduls
- **AT-Kommandos**
  Auszugsweise werden die AT-Befehle beschrieben, die im Zusammenhang mit dem Senden und Empfangen von Kurznachrichten notwendig sind.
- **Die Initialisierungsphase**
  beschreibt den Vorgang des Einbuchen des GSM-Gerätes ins Netz und Initialisierung des GSM-Gerätes aus der Sicht der Anwendung.
- **Aufbau einer *Protocol Data Unit*** (PDU)
  Die *Protocol Data Unit* ist die Datenstruktur, die von GSM-Geräten unmittelbar zum Senden und Empfangen von Kurznachrichten verwendet wird. Da wir hauptsächlich GPS-Positionsdaten – also Binärdaten – übertragen wollen, können wir das GSM-Modul nicht im *Textmodus* betreiben, sondern im sog. *PDU-Modus* und hier benötigen wir die genaue Kenntnis über den Aufbau einer *Protocol Data Unit.*

Datenübertragung

Die serielle Schnittstelle des FALCOM A1 ist werksseitig auf 9600 bps, 8 bit Daten, 1 Stopbit und keine Parität eingestellt.

Modemantworten beginnen immer mit <CR><LF> und enden mit <CR><LF>. Ist die Befehlssyntax falsch (oder der Befehl unbekannt) meldet das Gerät „ERROR". Ist die Syntax korrekt aber einige Parameter ungültig, sendet das Gerät „+CME ERROR: <err>" mit unterschiedlichen Error-Codes. Konnte ein Kommando ausgeführt werden, wird die Zeichenkette „OK" zurückgegeben.

Hinweis: Bei manchen Befehlen wie z.B. „AT+CPIN?" oder bei Meldung von eingehenden Ereignissen wird die Modemantwort nicht mit „OK" abgeschlossen.

AT-Kommandos

In diesem Abschnitt werden nur diejenigen Kommandos und Modemantworten beschrieben, die zum Senden und Empfangen von Short Messages notwendig sind. Für eine vollständige Beschreibung aller AT-Kommandos von GSM-Geräten sei folgendes Dokument verwiesen [GSM0707].

AT+CREG: Netzwerk Registrierung

Dieses Kommando wird benutzt um den Registrierungsstatus des GSM Gerätes feststellen zu können.

Festellen des Status´ der Netzwerkregistrierung:

```
AT+CREG?
```

Antwort vom GSM Modul:

```
+CREG: <mode>,<stat>
  <mode>
    0: Disable network registration unsolicited result code
    1: Enable network registration code result code +CREG: <stat>
  <stat>
    0: not registered, ME is not currently searching a new operator
    1: registered, home network
    2: not registered, ME currently searching a new operator to register to
    3: registration denied
    4: unknown
    5: registered, roaming
```

AT+CPIN: Eingabe von PIN/PUK-Code

Diese Kommando wird verwendet um die Gültigkeit des PIN- oder des PUK-Codes zu überprüfen. Ebenso läßt sich damit eine neue PIN definieren. Nach dem Einschalten des Gerätes oder einem Reset, muß der gültige PIN-Code angegeben werden, bevor es verwendet werden kann.

Eingabe der PIN:

```
AT+CPIN=<pin>
```

Antwort vom Gerät:

```
OK                wenn die PIN korrekt angegeben wurde
+CME ERROR: 16    „ungültiges Paßwort“
```

Nach drei erfolglosen Versuchen, muß die PUK eingegeben werden. Als zweiter Parameter zur PUK muß ein neuer PIN-Code angegeben werden:

Eingabe von PUK und neuer PIN:

```
AT+CPIN=<puk>,<new pin>
```

Antwort vom Gerät:

```
OK                wenn PUK korrekt war und PIN gesetzt werden konnte
+CME ERROR: 16    „ungültiges Paßwort“
```

Um herauszufinden, welcher Code (PIN oder PUK) eingegeben werden muß sendet eine Anwendung `AT+CPIN?` an das Gerät.

Mögliche Antworten vom Gerät:

```
+CPIN: READY      PIN wird nicht benötigt
+CPIN: SIM PIN    PIN ist erforderlich
+CPIN: SIM PUK    PUK ist erforderlich
+CME ERROR: <err> SIM-Fehler, oder SIM-Karte fehlt
```

Hinweis: Nach 10 fehlerhaften Eingaben der PUK wird die SIM-Karte gesperrt!

AT+CMGF: Setzen des Nachrichtenformates

Mit diesem Kommando kann entweder der Textmodus oder der sog. PDU[15]-Modus gesetzt werden. Im Textmodus werden alle Kommandos und Antworten im ASCII-Zeichensatz dargestellt.

Im PDU-Modus wird die gesamte SMS-Nachricht einschließlich der Header-Informationen in Form einer binären Zeichenkette übertragen.

Setzen des Nachrichtenformates:

```
AT+CMGF=0       setzt den PDU-Modus
AT+CMGF=1       setzt den Textmodus
```

Antwort vom Gerät:

```
OK              gesetzter Modus ist gültig
ERROR           Modus konnte nicht gesetzt werden
```

AT+CMGR: Nachricht lesen

Mit diesem Kommando liest eine Anwendung eingegangene Nachrichten. Das GSM-Gerät informiert die Anwendung mit `+CMTI` über das Eintreffen neuer Nachrichten. Eingeganene Nachrichten werden an einem Speicherplatz auf der SIM-Karte abgespeichert. Je nachdem ob der Textmodus oder der PDU-Modus eingestellt wurden,

[15] PDU, Abkürzung für *Protocol Data Unit.* Im Detail wird diese Datenstruktur im Abschnitt „Aufbau der Protocol Data Unit“ dieses Kapitels beschrieben.

werden die Nachrichten in ASCII-Test oder als Hexadezimalstring (PDU) ausgegeben.

Lesen einer Nachricht:

```
AT+CMGR=<index>    Liest die Nachricht an Speicherposition <index>
```

Antwort vom GSM-Gerät im Textmodus

```
+CMGR: "REC READ","+436764759093","26/01/99 14h41m43s"
Sein oder nicht sein, das ist hier die Frage!

OK
```

Antwort vom GSM-Gerät im PDU-Modus

```
+CMGR:<flag>,<length><CR><LF><pdu>
OK
```

wobei <flag> angibt ob die Nachricht schon ausgelesen wurde oder nicht (0 = record unread, 1 = record read), <length> ist die Anzahl der Zeichen der <pdu>, z.B.:

```
+CMGR: 0,26
040C9134764657093900F39950429144554808506 2B5452DCFE9
OK
```

AT+CMGL Nachrichten auflisten

Mit diesem Kommando kann eine Anwendung gespeicherte Nachrichten lesen. Als Parameter übergibt die Anwendung den Typ von Nachrichten, der aufgelistet werden soll:

Nachrichten auflisten:

```
AT+CMGL=<stat>
```

wobei <stat> den Status der Nachricht im Speicher angibt:

AT+CMGL=0	"REC UNREAD" – *empfangene ungelesen Nachrichten*
AT+CMGL=1	"REC READ" – *empfangene gelesene Nachrichten*
AT+CMGL=2	"STO UNSENT" – *gespeicherte nicht gesendete Nachrichten*
AT+CMGL=3	"STO SENT" – *gespeicherte gesendete Nachrichten*
AT+CMGL=4	"ALL" – *alle Nachrichten*

Antwort vom GSM-Gerät im Textmodus

```
+CMGL=<index>,<stat>,<da/oa>[,<alpha>,<scts>,<tooa/toda>,<length>]
<CR><LF><data>
```

Antwort vom GSM-Gerät im PDU-Modus

```
+CMGL=<index>,<stat>,<length<CR><LF><pdu>
```

Beispiele:

Anwendung → GSM	AT+CMGL=0	*neue ungelesene Nachrichten*
GSM → Anwendung	+CMGL: 1,"REC UNREAD","43322449"<CR> To be or not to be! +CMGL: 3,"REC UNREAD","46290800"<CR> Be happy! OK	

Anwendung → GSM	AT+CMGL=1	*gelesene Nachricht(en)*
GSM → Anwendung	+CMGL: 2, "REC READ","43322449",20<CR> Keep cool OK	
Anwendung → GSM	AT+CMGL=6	*ungültiger Index!*
GSM → Anwendung	+CMS ERROR: 321	*Fehler: ungültiger Index*

AT+CMGS: Nachricht senden

Mit diesem Kommando werden Nachrichten versandt. Je nach eingestelltem Nachrichtenformat (siehe +CMGF) müssen die Parameter des Kommandos unterschiedlich angegeben werden. <adress> bezeichnet die Adresse des Empfängers. Endezeichen ist ^Z (ASCII-Zeichen 26). Der Text kann alle existierenden Zeichen enthalten mit Ausnahme des ^Z Zeichens.

Nachricht senden im Textmodus:

```
AT+CMGS=<address><CR><ascii-text><^Z>
```

Im PDU-Modus muß zuerst die Länge der PDU (Protocol Data Unit) angegeben werden. Adresse, Text und andere Parameter sind in der PDU selbst enthalten.

Nachricht senden im PDU-Modus (Details dazu siehe Abschnitt „Aufbau einer PDU“):

```
AT+CMGS=<length><CR><pdu-hex-string><^Z>
```

Antwort des GSM-Gerätes im Erfolgsfall:

```
+CMGS : <mr>
OK
```

wobei <mr> eine Nachrichten-Referenznummer darstellt, die vom GSM Modul generiert wird. Sie beginnt mit 0 und wird mit jeder ausgehenden Nachricht (unabhängig davon ob sie erfolgreich gesendet werden konnte, oder ob ein Fehler aufgetreten ist) um eins erhöht. Sie hat einen Zyklus von 256 (d.h. auf 255 folgt 0).

Hinweis: Diese Nummer ist kein Speicherplatz-Index, ausgehende Nachrichten werden nicht gespeichert.

Beispiel zum Senden einer Nachricht im PDU Modus:

Anwendung → GSM	AT+CMGF=0	*setze PDU-Modus*
GSM → Anwendung	OK	*PDU-Modus OK*
Anwendung → GSM	AT+CMGS=14<CR>01F606912143 65000004C9E9340B<^Z>	*Sende Nachricht*
GSM → Anwendung	+CMGS: 246	
	OK	*Nachricht wurde gesendet*

AT+CMGD: Delete message

Das Kommando zum Löschen von Nachrichten sollte nach einem Read-Kommando verwendet werden, um die gespeicherte Nachricht zu löschen und damit den Speicherplatz für eine neue Nachricht freizugeben.

Nachricht löschen:

AT+CMGD=<index>

Antwort des GSM Moduls:

OK	<index> war ein gültiger Wert (Nachricht wurde glöscht)
ERROR	<index> war ungültig.

Initialisierungsphase

Bei der Initialisierung des GSM-Gerätes ist grundsätzlich zu prüfen, ob

(1) das Gerät angeschlossen ist,
(2) die PIN (oder PUK *und* PIN) eingegeben werden muß,
(3) das Gerät im Netz eingebucht ist.

Konnte das Gerät erfolgreich aktiviert werden (PIN/PUK wurde gesendet und Gerät ist eingebucht), wird mit dem Kommando +CMGF das gewünschte Nachrichtenformat eingestellt: Da auch binäre Daten (SMS-Requests, bzw. GPS-Positionen) übertragen werden, muß der sog. *PDU-Modus* eingestellt werden. Im PDU-Modus, wird eine SMS-Nachricht einschließlich aller Header-Informationen als binäre Zeichenkette übertragen.

Bei Aufruf der `gpsdev_Initialize()`-Funktion durch die GPSAPI.DLL sendet der Treiber zunächst das ATE0-Kommando (Echo aus) an das GSM-Modul. Erstens um unnötigen Datentransfer zu vermeiden und zweitens um festzustellen, ob das GSM-Modul eingeschaltet ist und auf Kommandos reagiert. Der weitere Verlauf der Initialisierungsphase ist dem umseitig angegebenen Zustandsdiagramm zu entnehmen.

Bei Zeitüberschreitung oder bei anderen als den im Diagramm angegebenen Antworten geht der GPS-Treiber in den ERROR-Zustand (diese Zustandsüberführungen sind nicht explizit angegeben). Der Treiber sendet im Fehlerfall eine entsprechende Fehlermeldung über die GPSAPI.DLL an die Anwendung. Die angegebenen Namen für die Zustände entsprechen den Konstanten, die im Sourcecode verwendet wurden.

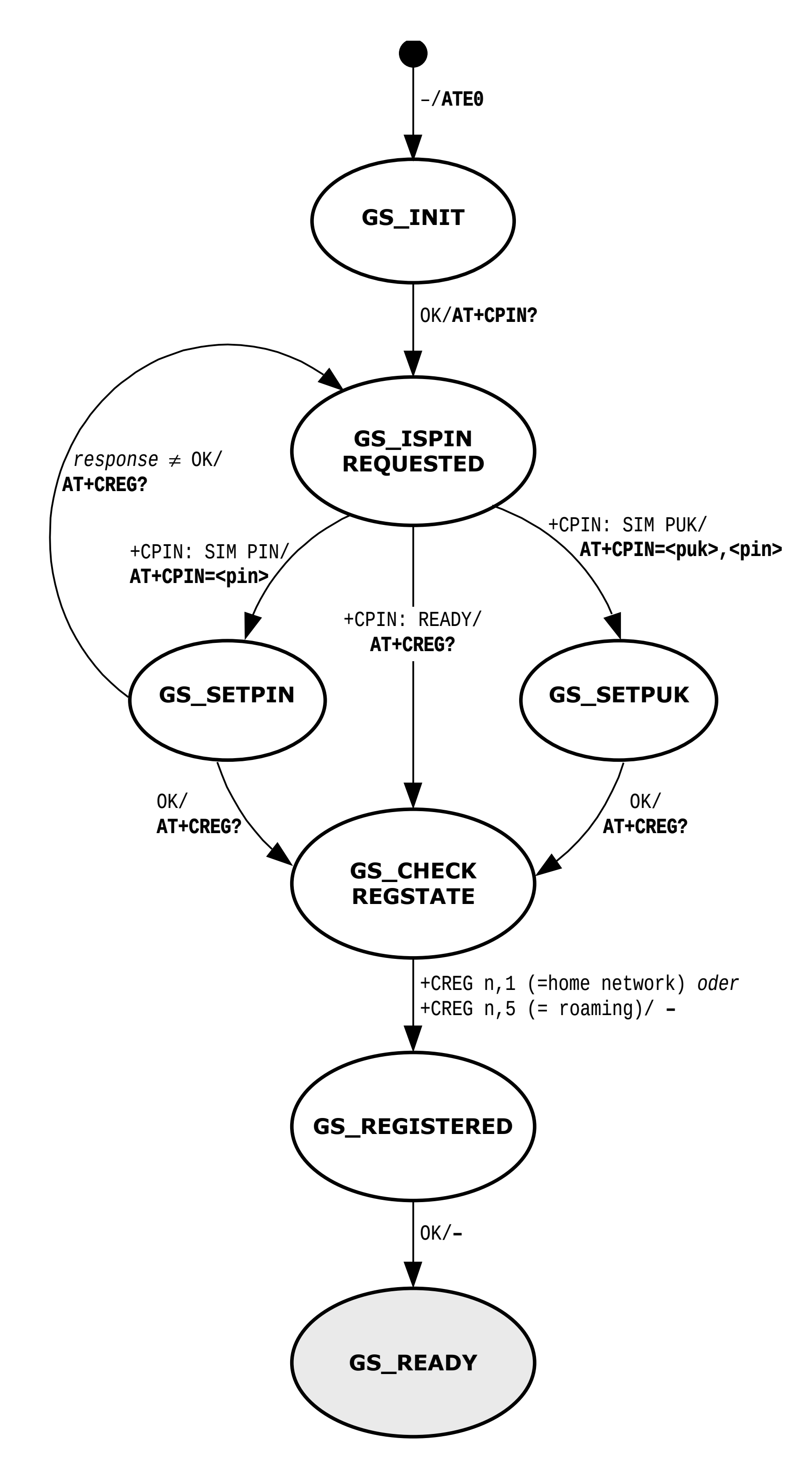


Abb. 25. Zustandsdiagramm der Initialisierungsphase

Hinweise:

- Tritt im Zustand GS_SETPUK ein Fehler auf (+CME ERROR: 16 = falsche PIN), so wird ein Fortsetzen der Initialisierung nicht zum Ziel führen. Nach 10 Fehlversuchen würde die SIM-Karte gesperrt. Der Gerätetreiber geht daher in den Fehlerzustand über und gibt eine entsprechende Warnung an den Benutzer aus („PUK ungültig oder nicht angegeben!"). Die Fehlerursache kann der Ereignisanzeige entnommen werden.

- Bestimmte „Antworten" können immer hereinkommen, unter Umständen auch schon während der Initialisierungsphase (z.B.: RING, +CMTI, ...). Diese Nachrichten werden in der `gpsdev_Decode()`-Routine daher gesondert behandelt: Es wird zunächst nur ein Flag gesetzt, das das Auftreten jedes dieser Ereignisse anzeigt. Behandelt werden die Antworten erst, wenn sich der Gerätetreiber im Zustand *GS_READY* befindet.

Aufbau einer *Protocol Data Unit* (PDU)

Innerhalb der Short-Message-Transportschicht werden Kurznachrichten in Form von *Protocol Data Units (PDUs)* übertragen. Es gibt sechs verschiedene Typen von PDUs:

- SMS-SUBMIT, überträgt eine SMS von der MS (Mobilstation) an das SMSC (Short Message Service Center)
- SMS-DELIVER, überträgt eine Kurznachricht vom SMSC zur MS
- SMS-DELIVER-REPORT, übermittelt Fehlerursachen (SMSC an MS)
- SMS-SUBMIT-REPORT, übermittelt Übertragungsfehler (MS an SMSC)
- SMS-STATUS-REPORT, überträgt Status-Meldungen vom SMSC an die MS
- SMS-COMMAND, sendet ein Kommando von der MS an das SMSC

In unserem Fall sind nur die ersten beiden Typen (SMS-SUBMIT und SMS-DELIVER) also Versand und Empfang von Nachrichten von einer Mobilstation von Interesse.

Die folgenden beiden Skizzen zeigt den schematischen Aufbau dieser PDUs[16]:

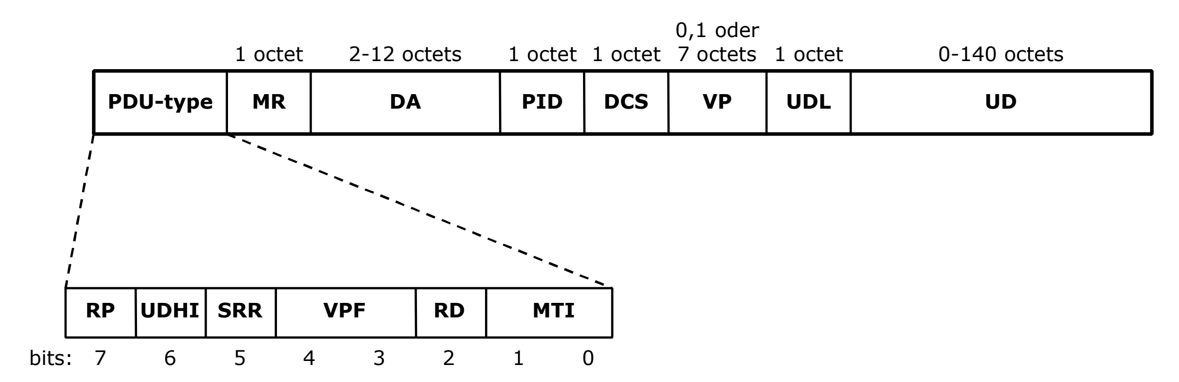


Abb. 26. Aufbau der PDU einer ausgehenden Nachricht (SMS-SUBMIT)

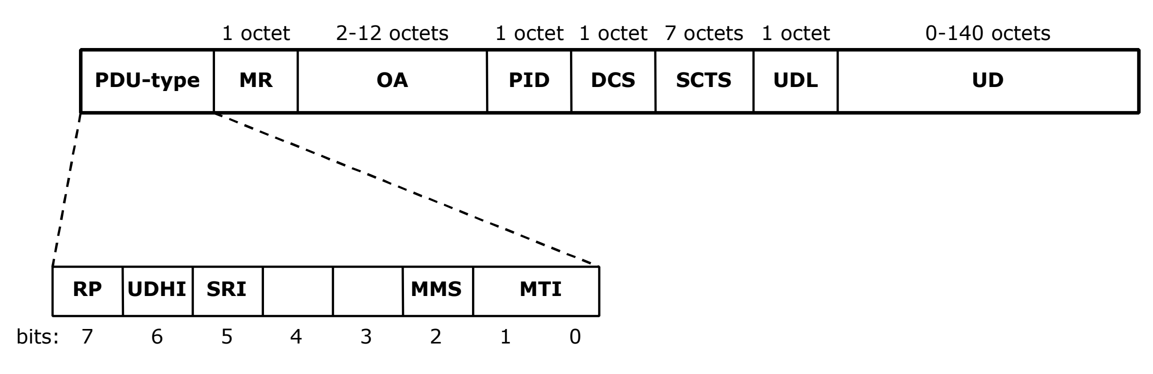


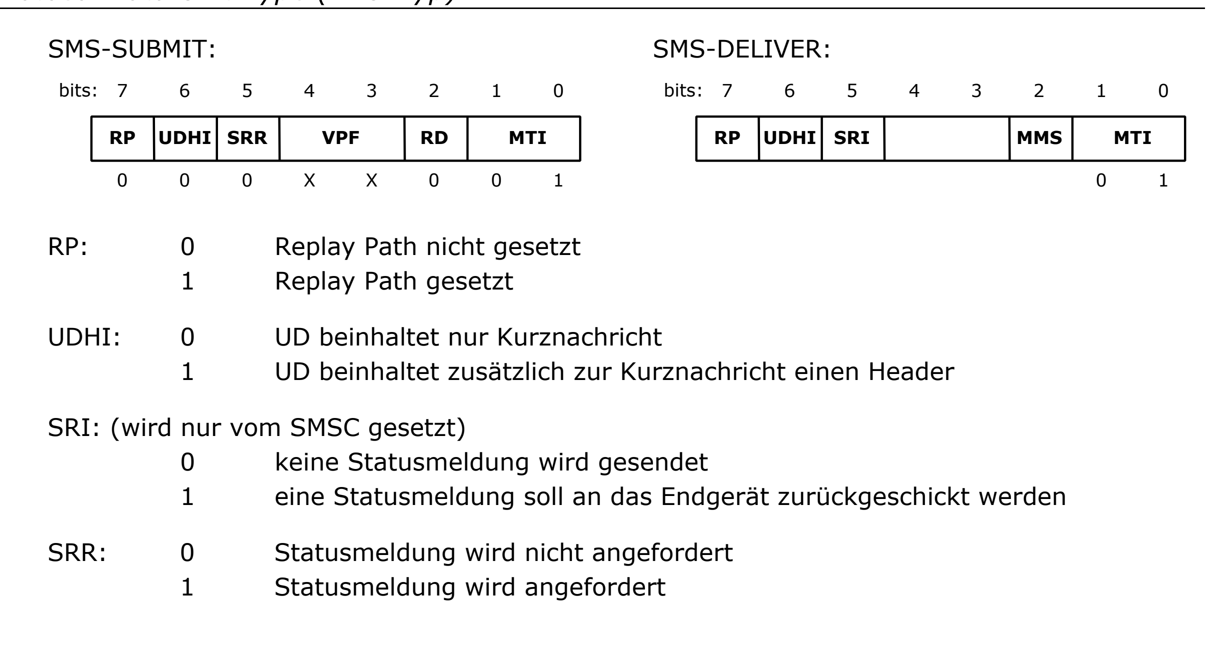
Abb. 27. Aufbau der PDU einer eingehenden Nachricht (SMS-DELIVER)

[16] In Anlehnung an die ETSI GSM Spezifikationen wird hier der Begriff Oktett (oder *octet*) verwendet. Bezeichnet wird damit ein Datentyp mit 8 Bit.

Die Bedeutung der einzelnen Datenfelder:

Abk.		**Bezeichnung**	**Beschreibung**
PDU-Typ	RP	*Reply Path*	Parameter gibt an, ob Absenderadresse existiert
PDU-Typ	UDHI	*User Data Header Indicator*	Parameter gibt an, ob die Benutzerdaten einen Header enthalten
PDU-Typ	SRR	*Status Report Request*	Parameter gibt an, ob die MS eine Statusmeldung angefordert hat
PDU-Typ	SRI	*Status Report Indicator*	Parameter gibt an, ob die Gegenstation eine Statusmeldung angefordert nat
PDU-Typ	VPF	*Validity Period Format*	Parameter gibt an, ob das VP-Feld (Validity Period) angegeben ist
PDU-Typ	RD	*Reject Duplicate*	wenn gesetzt, weise das SMSC Nachrichten mit gleicher MR und DA zurück
PDU-Typ	MMS	*More Messages to Send*	Parameter gibt an, ob noch weitere Nachrichten zum Senden vorhanden sind
PDU-Typ	MTI	*Message Type Indicator*	Parameter bestimmt den Nachrichten-Typ: 00 = SMS-DELIVER 01 = SMS-SUBMIT
MR		*Message Reference*	fortlaufende Referenznummer (0 .. 255, zyklisch); wird vom Gerät gesetzt.
DA		*Destination Adress*	Zieladresse
OA		*Originator Adress*	Absenderadresse
PID		*Protocol Identifier*	Parameter, der dem SMSC anzeigt, wie die Nachricht zu behandeln ist (als FAX, Voice, SMS usw.)
DCS		*Data Coding Scheme*	Parameter gibt an, wie die Nutzdaten (User Data) codiert sind (7-bit oder 8-bit)
SCTS		*Service Center Time Stamp*	Parameter gibt Zeitpunkt des Eintreffens der Nachricht im SMSC an
VP		*Validity Period*	Parameter gibt die Dauer an, wie lange die Nachricht vom SMSC gespeichert werden muß
UDL		*User Data Length*	Anzahl der Zeichen der Kurznachricht
UD		*User Data*	Daten der Kurznachricht

Protocol Data Unit Type (PDU-Typ):



RP: 0 Replay Path nicht gesetzt
1 Replay Path gesetzt

UDHI: 0 UD beinhaltet nur Kurznachricht
1 UD beinhaltet zusätzlich zur Kurznachricht einen Header

SRI: (wird nur vom SMSC gesetzt)
0 keine Statusmeldung wird gesendet
1 eine Statusmeldung soll an das Endgerät zurückgeschickt werden

SRR: 0 Statusmeldung wird nicht angefordert
1 Statusmeldung wird angefordert

VPF:	bit4	bit3	
	0	0	Validity Period-Feld nicht vorhanden
	0	1	reserviert
	1	0	VP Feld vorhanden (relative Dauer)
	1	1	VP Feld vorhanden (absolute Zeitangabe)

MMS: (wird nur vom SMSC gesetzt)

0 Im SMSC wind weitere Nachrichten für diese MS vorhanden

1 Keine weiteren Nachrichten vorhanden

RD: 0 Weist das SMSC an eine Nachricht zu akzeptieren, die dieselbe MR und DA besitzt als eine zuvor gesendete, und noch nicht zugestellte Nachricht.

1 Weist das SMSC an Nachrichten mit gleicher MR und DA zurückzuweisen (sofern noch vergleichbare Nachrichte im SMSC vorliegen)

MTI:	bit1	bit0	Message type
	0	0	SMS-DELIVER (SMSC → MS)
	0	0	SMS-DELIVER REPORT (MS → SMSC)
	0	1	SMS-SUBMIT (MS → SMSC)
	0	1	SMS-SUBMIT REPORT (SMSC → MS)
	1	0	SMS-STATUS REPORT (SMSC → MS)
	1	0	SMS-COMMAND (MS → SMSC)
	1	1	Reserved

Message Reference MR

Die MR ist eine fortlaufende Referenznummer (0..255) einer SMS-SUBMIT-PDU. Sie wird vom GSM-Modul automatisch gesetzt.

Originator Adress OA und *Destination Adress DA*

OA und DA haben dasselbe Format, das wie folgt definiert ist:

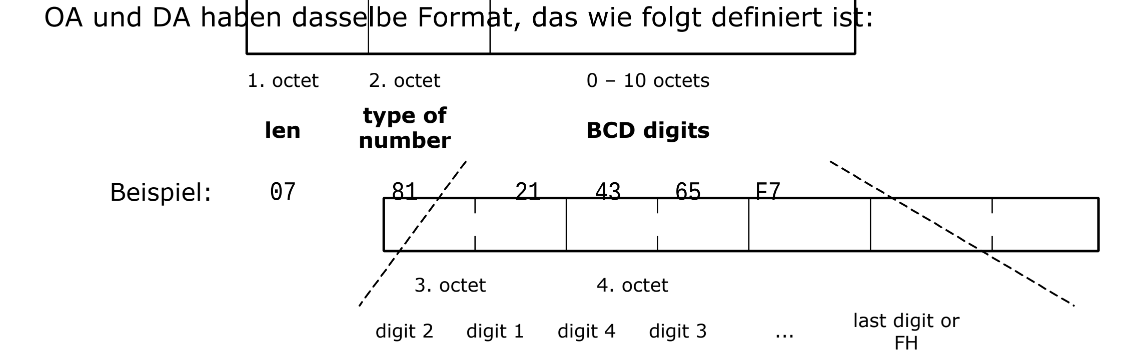


Abb. 28. Format von OA/DA

len: Anzahl der BCD-Ziffern der Adresse (nicht die Anzahl der Octets!)

type of number: 81H – die Nummer hat nationales Format (0732...)
91H – die Nummer hat internationales Format (+43676...)

BCD digits: enthält die Nummer des Empfängers/Absenders im BCD-Format. Ist die Anzahl der Ziffern ungerade, muß die letzte Ziffer mit hex F aufgefüllt werden. Zusätzlich müssen Low- und High-Nibbel vertauscht werden („12 34" wird codiert als „21 43").

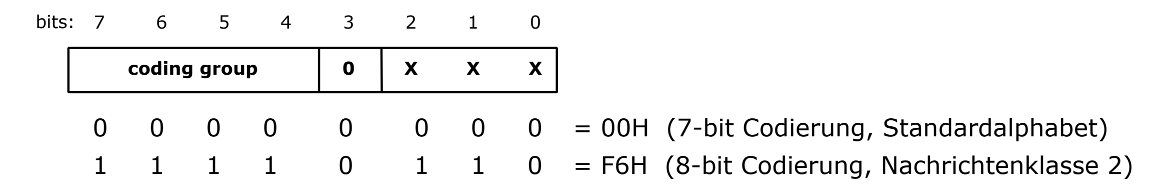
Beispiel: Die Nummer 1234567 führt zur *Destination Address:* „0781214365F7".

Protocol Identifier PID

Die PID gibt an wie die Nachricht im SMSC behandelt werden soll. Für unseren Zweck benötigen wir nur PID = 00H (PDU ist eine Short Message). Detailinformationen siehe [GSM0340], Kapitel 9.2.3.9.

Data Coding Scheme DCS

Das DCS Feld spezifiziert das Datencodierschema der Nutzdaten (User Data), verwendetes Alphabet und Nachrichtenklasse.



Die Bits 7..4 spezifizieren das Codierungsschema. Die Bits 3..0 werden in Abhängigkeit davon wie folgt verwendet:

coding group bits 7..4	**bits 3..0**
0000	*Alphabet:* 0000 Default Alphabet (7 bit Codierung in UD) 0001-1111 reserviert
0001-1110	reservierte Codierungsgruppen
1111	*Datencodierung/Nachrichtenklasse:* bit 3: reserviert, auf 0 gesetzt bit 2: *Datencodierung:* 0 Default alphabet (7 bit data coding in the User Data) 1 8-bit Datencodierung bit 1: bit 0: *Nachrichtenklasse:* 0 0 Class0 immediate display 0 1 Class1 ME (Mobile Equipment)-specific 1 0 Class2 SIM specific message 1 1 Class3 TE (Terminate Equipment)-specific

7-Bit-Alphabet

Wird das Standardalphabet verwendet, so werden die Nutzdaten (User Data) im 7-bit Alphabet[17] codiert. Wird dieses Alphabet verwendet, kann die Nachricht aus bis zu 160 Zeichen (statt 140 Zeichen in 8-bit-Codierung) bestehen.

Nachrichtenklassen

Die Nachrichtenklassen spezifizieren wie die Nachricht behandelt werden soll: In *Class 0* wird die Nachricht sofort auf dem Display dargestellt, in *Class 1* im ME (Mobile Equipment) also im Speicher des Gerätes selbst abgelegt, in *Class 2* auf der SIM-Karte gepsichert und bei *Class 3* an ein TE (Terminal Equipment), an ein angeschlossenes Gerät ausgegeben. Diese Option für Nachrichtenklassen muß allerdings nicht in allen GSM-Geräten implementiert sein.

[17] Alphabet und 7-Bit-Codierung sind definiert in: ETSI GSM 03.38: Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information

Service Center Time Stamp SCTS

Die SCTS informiert den Empfänger darüber, wann die Nachricht im SMSC eingetroffen ist. Die Zeitangabe bezieht sich auf Lokalzeit.



Beispiel: 9 9 5 0 1 2 3 1 5 4 1 3 0 0

bedeutet: 21. Mai 1999, 13:45:31 Uhr

Die Zeitzone gibt die Differenz (Anzahl der Viertelstunden) zwischen Lokalzeit und GMT an.

Validity Period VP

Das VP-Feld bestimmt die Gültigkeitsdauer einer Kurznachricht. Konnte die Nachricht vom SMSC nicht innerhalb der angegebenen Zeitspanne zugestellt werden, kann sie vom SMSC gelöscht werden. Es existieren zwei Möglichkeiten die *Validity Period* festzulgen: *relativ* oder *absolut*.

Im ersten Fall (relative Zeitangabe) besteht das VP-Feld nur aus einem *Octet*. Es ist ein Wert der angibt wie lange die Nachricht gültig ist, beginnend mit dem Zeitpunkt ihres Eintreffens im SMSC. Der VP-Wert ist wie folgt definiert:

VP Value	*tatsächliche Gültigkeitsdauer*
0-143	(VP + 1) x 5 Minuten (also 5 Minuten Intervalle bis 12 Stunden)
144-167	12 Stunden + ((VP-143) x 30 Minuten)
168-196	(VP - 166) x 1 Tag
197-255	(VP - 192) x 1 Woche

Im zweiten Fall ist die Definition identisch zur Definition der SCTS (*Service Center Time Stamp*). Welcher der beiden Arten verwendet wird ist im VPF-Feld (*Validity Period Format*) des PDU-Typs angegeben.

User Data Length UDL and User Data UD

Das UDL Feld gibt die Anzahl der Zeichen in der Nachricht an (nicht die Anzahl der Octets im UD-Feld!)

1 octet 0 .. 140 octets

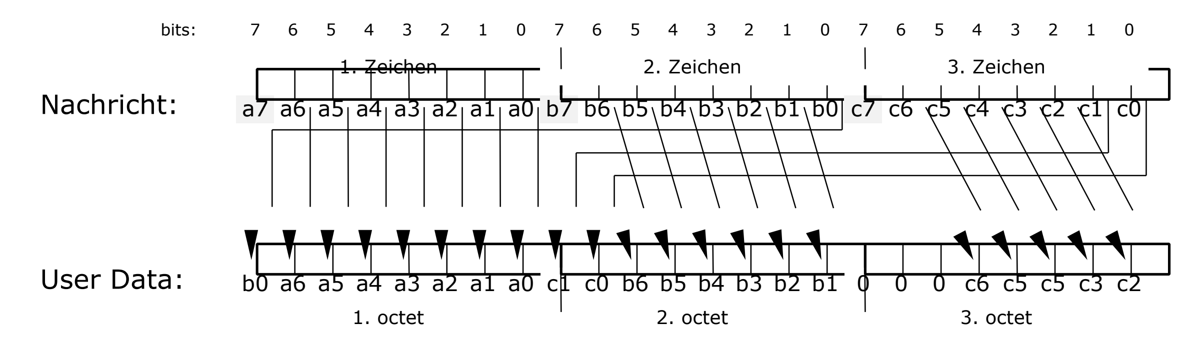
UDL User Data

Beispiel: 05 E8 32 9B FD 06

bedeutet: „hello"im Standardalphabet

Das UD-Feld enthält nun die eigentliche Nachricht. Wie bereits erwähnt, können die Daten 8-bit oder 7-bit codiert sein. Bei 8-Bit-Codierung können die Daten einfach als ASCII-Zeichen interpretiert werden.

Soll die 7-Bit-Codierung (für Textnachrichten) verwendet werden, so sind die Daten so zu packen, wie in [GSM0338] definiert. Die folgende Skizze versucht am Beispiel der Codierung von 3 Zeichen die 7-Bit-Codierung zu veranschaulichen:



Auf diese Weise ist es möglich eine Nachricht mit 160 Zeichen (GSM Stanardalphabet) in 140 Bytes zu packen. Nicht verwendete Bits werden auf 0 gesetzt.

Anhang: Tabelle des GSM Standardalphabets

Dieses Alphabet muß in jedem GSM-Gerät implementiert sein. Weitere Alphabete sind optional.

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7
0	0	0	0	0	@	Δ	SP	0	¡	P	¿	p
0	0	0	1	1	£	1)	!	1	A	Q	a	q
0	0	1	0	2	$	Φ	"	2	B	R	b	r
0	0	1	1	3	¥	Γ	#	3	C	S	c	s
0	1	0	0	4	è	Λ	¤	4	D	T	d	t
0	1	0	1	5	é	Ω	%	5	E	U	e	u
0	1	1	0	6	ù	Π	&	6	F	V	f	v
0	1	1	1	7	ì	Ψ	'	7	G	W	g	w
1	0	0	0	8	ò	Σ	(	8	H	X	h	x
1	0	0	1	9	Ç	Θ	)	9	I	Y	i	y
1	0	1	0	10	LF	Ξ	*	:	J	Z	j	z
1	0	1	1	11	Ø	1)	+	;	K	Ä	k	ä
1	1	0	0	12	ø	Æ	,	<	L	Ö	l	ö
1	1	0	1	13	CR	æ	-	=	M	Ñ	m	ñ
1	1	1	0	14	Å	ß	.	>	N	Ü	n	ü
1	1	1	1	15	å	É	/	?	O	§	o	à

1) Diese Zeichen sind nicht definiert, sollten aber als Leerzeichen dargestellt werden.

Dateireferenz TARGET.GPS

Folgende Dateien sind dem Target-(GSM)-Gerätreiber zugeordnet:

Datei	**Beschreibung**
Verzeichnis *Source:*	
GpsDev.h	Headerdatei für alle GPS-Gerätetreiber
Target.h/.cpp	Implementierung der gerätespezifischen Funktionen
DSmsOpt.h/.cpp	Target Setup Dialog (in erster Linie Einstellungen für GSM)
fill.h/.cpp	enthält eine Hilfsfunktion zum Füllen der GPSAPI_POS-Struktur
Target.def	Projektdefinitionsdatei
Verzeichnis *Res:*	
targ_gr.rh/.rc	Resource-Dateien (deutsch)
targ_bmp.rc	Icons und Bitmaps
Verzeichnis *misc/Source:*	
log.h/.cpp	Logfile-Funktionen (für Debugging Zwecke)
Verzeichnis *.\exe:*	
GpsAPI.lib	Bibliothek zum Linken des GpsAPI's (Loadtime-Linking)

Kapitel 5

Dokumentation MapAPI

Dieses Kapitel dokumentiert das MAP-Application Programming Interface (MAPAPI), eine DLL zur Verwaltung und Darstellung von Karten (gescannte Bitmaps) und geographischen Informationen (GPS-Positionen, Vektordaten).

Einleitung

Das MAPAPI bitetet dem Benutzer eine Schnittstelle zur Verwaltung von Karten und Layern, das es ihm ermöglicht, auf einfache Weise Karten, Layer und andere geographische Objekte darzustellen, ohne sich mit Details der Speicherung und Darstellung von geographischen Daten befassen zu müssen.

Ebenso braucht der Benutzer sich nicht um die Umrechnung der Koordinaten in das verwendete Koordinatensystem zu kümmen. Das MAPAPI bietet hier ausreichende Unterstützung, um beliebige Koordinatensysteme für eine bestimmte Karte auszuwählen. Die Koordinaten aller Objekte, die dann in diese Karte eingefügt werden, werden automatisch vom MAPAPI in dieses Koordinatensystem transformiert.

Es wurde hier bereits mehrmals der Begriff „Layer“ verwendet. Damit werden die verschiedenen Ebenen einer Vektorkarte bezeichnet. Es können beispielsweise Straßen, Flüsse, Seen, Radwege, etc. in verschiedenen Ebenen liegen. Im GPSAPI wird jedes Objekt als eigene Ebene (Layer) behandelt.

Das MAPAPI erlaubt es Pixelkarten mit Vektordaten zu kombinieren. Eine Pixelkarte wird dann ebenfalls als Layer betrachtet.

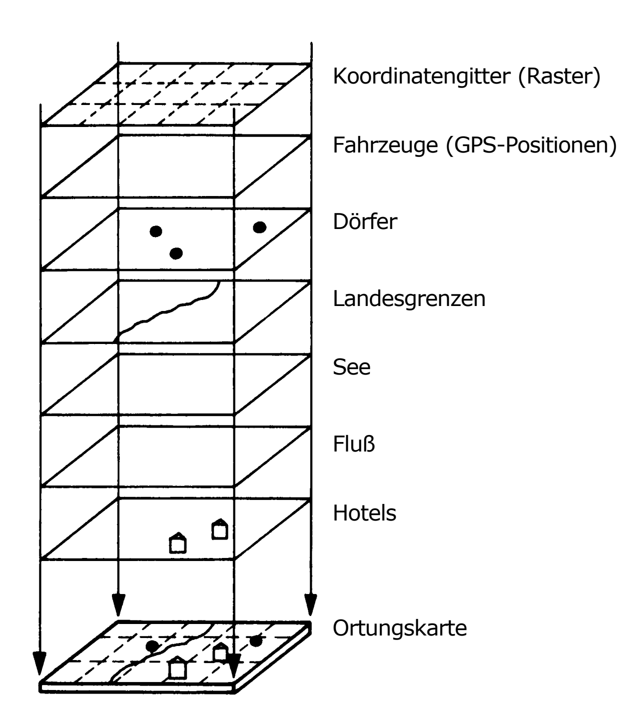


Abb. 29. Vektorkarten können aus mehreren Ebenen aufgebaut werden.

Komponenten

Zentraler Oberbegriff des MAPAPI’s ist eine *Karte* („MAP“). Das API erlaubt einer Anwendung, belieibg viele Karten (Maps) zu verwalten. Eine Karte definiert das zu verwendende Koordinatensystem, den Zoomfaktor und Ränder für eine bestimmte Anzahl von darzustellenden Objekten.

Alle auf einer Karte darstellbaren Objekte werden als Layer bezeichnet. Das können Bitmaps (gescannte Grafiken) Vektordaten (Menge von GPS-Positionen eines Fahrzeuges) oder Punktobjekte (Referenzpunkte, Standorte von Objekten) sein.

Aus Gründen der besseren Handhabbarkeit und Übersichtlichkeit werden Layer zu Gruppen zusammengefaßt. Bevor ein Layer in eine Karte eingefügt werden kann, muß eine Gruppe in der Karte definiert werden.

Objektmodell

Klasse TMapFrame

Rahmenfenster für Karte, Statusfenster (am unteren Rand) und Layer-Baumansicht (linker Fensterbereich)

Klasse TMap

TMap ist direkt vom OWL-Objekt TWindow abgeleitet, und erbt daher alle Möglichkeiten der Ausgabe von Daten im TWindow-Objekt. TMap speichert alle statischen (Kartengröße, Größe des Ausschnittes, aktuelles Koordinatensystem) und dynamischen Parameter (aktueller Zoomfaktor). Es enthält eine Liste von Gruppen (TGroupList).

Klasse TGroup

Enthält eine Liste von Layern.

Klasse TLayer

TLayer ist die abstrakte Basisklasse für alle weiteren Layer (sowohl Pixel-Layer als auch Vektorlayer) und definiert dadurch die Schnittstelle für einen Layer.

Klasse TVectLayer

Basisklasse für alle Vektorlayer (TCorLayer und TDXFLayer). Wird auch für neue GPS-Fahrzeug-Positionen verwendet.

Klasse TBitmapLayer

Klasse für Pixel-Karten, die in „einem“ Stück vorliegen (also ein großes Bitmap)

Klasse TBmpPieceLayer

Klasse für Pixel-Karten die „gestückelt“ vorliegen (viele Kacheln von Bitmaps)

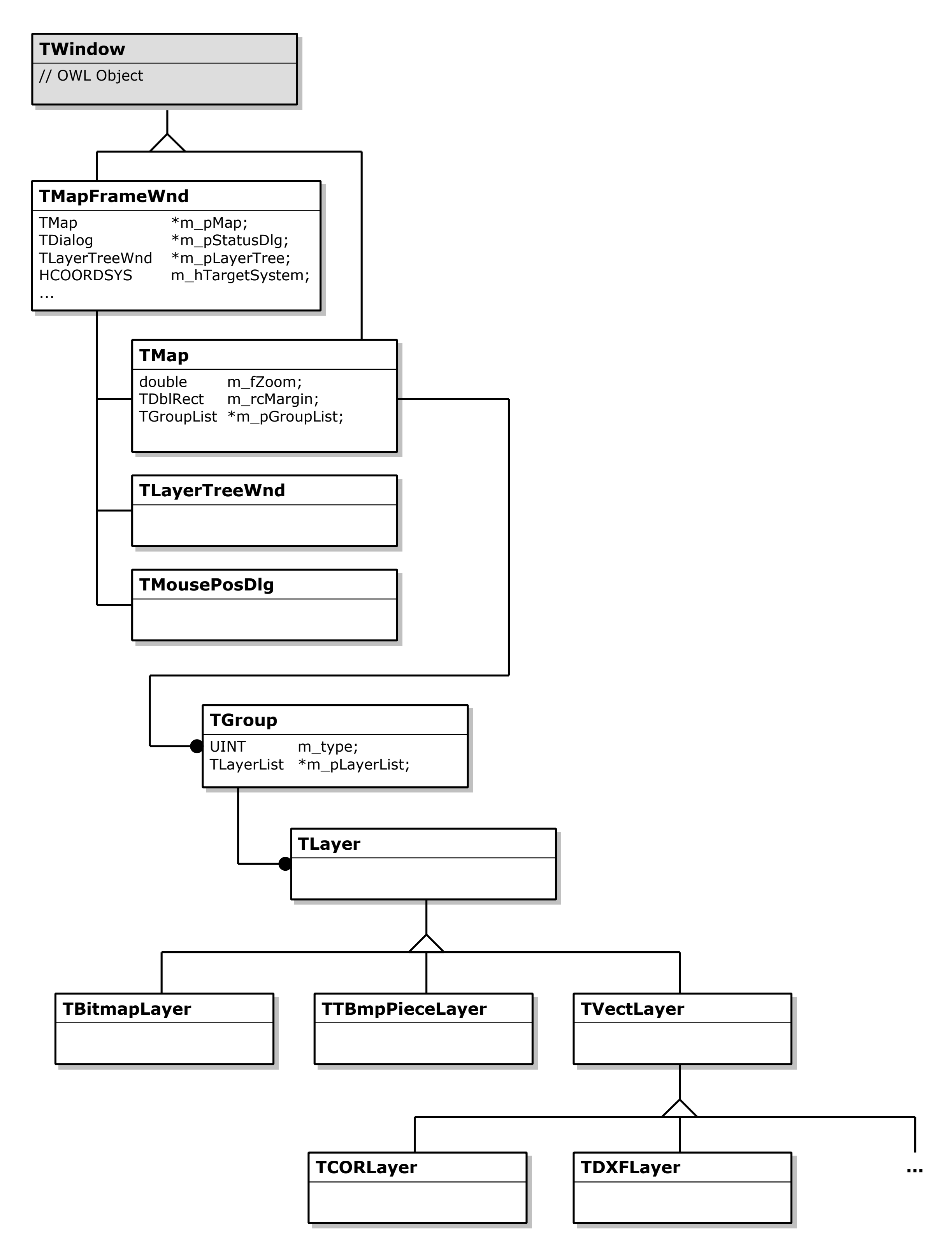


Abb.30. Das MAPAPI-Objektmodell

Schnittstelle

Dieses Kapitel enthält die Beschreibung des Map Aplication Programming Interfaces.

HANDLE mapapi_OpenMap (HWND hwndOwner,
RECT* prcSize,
DWORD mapStyle);

Die Funktion **mapapi_OpenMap()** öffnet eine neue, leere Karte (eigentlich ein Karten*fenster*) innerhalb des durch *hwndOwner* angegebenen Fensters mit einer bestimmten Größe. Mit **mapapi_ResizeMap()** kann die Größe nachträglich jederzeit verändert werden. Es werden bzgl. Hintergrundfarbe, Raster, Zoomfaktor, etc... bestimmte Standardwerte verwendet.

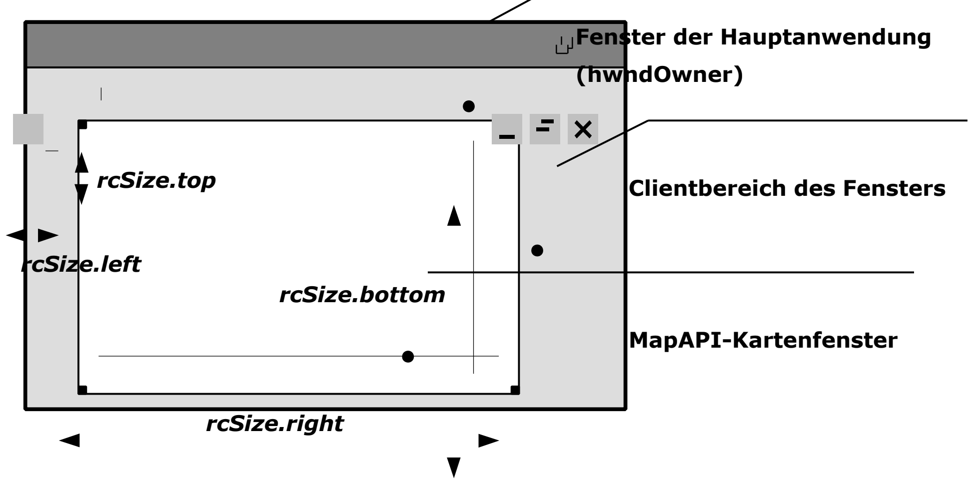
Parameter:

HWND *hwndOwner*

Fensterhandle des Fensters innerhalb dem die Karte angezeigt wird.

RECT *rcSize*

Anfangsgröße und Position des Kartenausschnittes in Pixel relativ zum Clientbereich des durch *hwndOwner* angegebenen Fensters.



DWORD *mapStyle*

Dieser Parameter legt bestimmte Kartenstile fest. Eine Kombination folgender Werte ist möglich:

Kartenstil	Beschreibung
MS_VISIBLE	Karte ist sichtbar (analog zum Windows-Fenstertil WS_VISIBLE).
MS_SHOWGRID	Auf der Karte wird ein Raster dargestellt.
MS_SHOWSTATUS	Blendet im unteren Bereich des Kartenfensters eine Statuszeile ein, innerhalb der die Rastergröße, eine Maßstabsleiste, sowie die aktuelle Mausposition relativ zur Karte (geographische Länge und Breite) angezeigt werden.

MS_LAYERLIST	Öffnet im linken Fensterbereich eine (explorerähnliche) Baumansicht, in der die auf der Karte dargestellten Layer aufgelistet werden.

Rückgabewert:

Handle des neuen Fensters. Dieses Handle wird bei allen weitern Funktionsaufrufen benötigt, die sich auf diese Karte beziehen. Der Rückgabewert ist 0, falls beim Erzeugen der Karte ein Fehler aufgetreten ist.

BOOL mapapi_CloseMap (HANDLE hMap);

Schließt die angebene Karte und gibt alle Resourcen frei, die von den innerhalb der Karte dargestellten Layerobjekten belegt wurden.

Parameter:

HANDLE *hMap*
Handle der Karte die geschlossen werden soll.

Rückgabewert:
TRUE bei Erfolg, FALSE sonst.

DWORD mapapi_GetMapStyle (HANDLE hMap);

mapapi_GetMapStyle() ermittelt den aktuellen Kartenstil der angegebenen Karte.

Parameter:

HANDLE *hMap*
Handle der Karte.

Rückgabewert:
Der Rückgabewert ist eine Kombination der MS_xxx-Konstanten

DWORD mapapi_SetMapStyle (HANDLE hMap, DWORD mapStyle);

Setzt einen neuen Kartenstil für die angegebene Karte. Diese Funktion ermöglicht es einer Anwendung Raster, Statusleiste und Layerfenster ein- und auszublenden, bzw. die gesamte Karte zu verbergen oder anzuzeigen.

Parameter:

HANDLE *hMap*
Handle der Karte.

DWORD *mapStyle*

Dieser Parameter legt den neuen Kartenstil fest. Es ist eine beliebige Kombination der MS_XXX-Konstanten möglich (siehe dazu **mapapi_OpenMap()**).

Rückgabewert:

Der Rückgabewert ist der vorherige Kartenstil.

---

void mapapi_ResizeMap (HANDLE hMap, RECT *prcNewSize);

Ändert die Größe des Kartenfensters. Diese Funktion muß von der Anwendung, die das Kartenfenster besitzt, aufgerufen werden, wenn sich Größe des Fensters ändert, innerhalb dessen die Karte dargestellt wird. Dies ist z.B. dann notwendig, wenn die Karte innerhalb eines MDI-Kindfensters dargestellt wird und immer dessen gesamten Client-Bereich ausfüllen soll.

Parameter:

HANDLE *hMap*

Handle der Karte.

RECT **prcNewSize*

Zeiger auf eine RECT-Struktur mit den neuen Koordinaten (relativ zum Clientbereich des Fensters, in dem die Karte angezeigt wird).

Rückgabewert:

keiner.

---

void mapapi_SetFocus (HANDLE hMap);

Die **mapapi_SetFocus()** Funktion setzt den Eingabefocus auf die angegebene Karte. Alle Tastaturnachrichten werden dann direkt an die Karte geleitet.

Parameter:

HANDLE *hMap*

Handle der Karte.

Rückgabewert:

keiner.

HANDLE mapapi_AddGroup (HANDLE hMap, LPSTR lpszName, UINT type);

Erzeugt in der angegebenen Karte eine neue Layergruppe.

Parameter:

HANDLE *hMap*
Handle der Karte.

LPSTR *lpszName*
Name der anzulegenden Gruppe.

UINT *type*
Typ der Gruppe. Diese Typisierung gibt Auskunft darüber, welche Daten in diesem Layer eingefügt werden sollten. Grundsätzlich ist es jedoch möglich einen Layer einer beliebigen Gruppe zuzuordnen. Folgende Gruppentypen sind verfügbar:

Typ	Beschreibung
GT_MAP	Layer dieser Gruppe, sind üblicherweise Bitmaps (z.B. gescanntes Kartenmaterial)
GT_STATIC	Layer dieser Gruppe beinhalten Vektordaten, es kommen aber keine weiteren GPS-Positionen mehr hinzu, sie sind also „statisch" (z.B. mit GPS-Empfänger vermessene Objekte)
GT_DYNAMIC	Diese Gruppe ist für Fahrzeuglayer vorgesehen, bzw. für Layer die laufend um neue GPS-Positionen erweitert werden.
GT_POINT	Layer dieser Gruppe beinhalten nur eine einzige GPS-Position (Referenzpunkte, Standorte von bestimmten Objekten, ...)

Rückgabewert:
Gruppen-Handle, das die Gruppe eindeutig identifiziert. Es wird dazu benötigt, wenn in der Folge Layer in die Karte eingefügt werden sollen.

HANDLE mapapi_GetFirstGroup (HANDLE hMap);

Liefert die erste Gruppe der angegebenen Karte.

Parameter:

HANDLE *hMap*
Handle der Karte.

Rückgabewert:
Gruppen-Handle. Der Rückgabewert ist NULL, wenn die Karte keine Gruppen enthält.

HANDLE mapapi_GetNextGroup (HANDLE hMap);

Liefert in aufeinanderfolgenden Aufrufen jeweils die nächste Gruppe der Karte.

Parameter:

HANDLE *hMap*
Handle der Karte

Rückgabewert:
Gruppen-Handle. Der Rückgabewert ist NULL, wenn die Karte keine weiteren Gruppen mehr enthält.

BOOL mapapi_GetGroupData (HANDLE hGroup, GROUP_DATA* pData);

Die Funktion **mapapi_GetGroupData()** ermittelt die Eigenschaften einer bestimmten Gruppe.

Parameter:

HANDLE *hGroup*
Handle einer bestimmten Gruppe.

GROUP_DATA* pData
Zeiger auf eine GROUP_DATA Datenstruktur, die in Abhängigkeit des *mask*-Datenfeldes gefüllt wird:

```
typedef struct {
  UINT   mask;          // a set of bit flags that specify which data members
                        //   are to be used
  LPSTR  pszText;       // name of this group
  UINT   type;          // group type
  LPARAM lParam;        // 32bit value associated with this group
} GROUP_DATA;
```

mask kann eine beliebige Kombination folgender Werte sein:

Gruppenflag	Beschreibung
GF_TEXT	Der Name der Gruppe soll zurückgegeben werden. *pszText* muß in diesem Fall auf einen genügend großen Puffer weisen. Die maximale Länge eines Gruppennamens ist mit MAX_PATH definiert.
GF_TYPE	*type* soll beim Funktionsaufruf gesetzt werden
GF_PARAM	*lParam* soll beim Funktionsaufruf gesetzt werden

Rückgabewert:
TRUE bei Erfolg, FALSE sonst.

BOOL mapapi_AddGroupMenuItem (HANDLE hGroup,
HWND hwndOwner,
int cmdId,
const char* lpszCommand,
DWORD dwFlags);

Mit Hilfe dieser Funktion kann einer bestimmten Gruppe ein Menüeintrag zugeordnet werden, der angezeigt wird, wenn der Benutzer mit der rechten Maustaste auf die Gruppe oder auf einen Layer klickt, der zu dieser Gruppe gehört.

Parameter:

HANDLE *hGroup*
Handle einer bestimmten Gruppe.

HWND *hwndOwner*
Fensterhandle des Fensters, an das der Menübefehl zur Auswertung gesendet werden soll.

int *cmdId*
ID des Menüeintrages.

const char **lpszCommand*
Text des Menüeintrages.

DWORD *dwFlags*
Flags, die bestimmen, wie der Menüeintrag angezeigt (enabled/disabled) werden soll. Eine Kombination der folgenden Werte ist möglich:

Menü-Flag	Beschreibung
GMI_GROUP	Der Menüeintrag ist genau dann verfügbar (enabled) wenn die Gruppe selbst markiert ist.
GMI_LAYER	Der Menüeintrag ist genau dann verfügbar, wenn ein Layer der Gruppe markiert ist.

Werden beide Flags gesetzt ist der Menüeintrag natürlich immer verfügbar.

Rückgabewert:
TRUE bei Erfolg, FALSE sonst.

HANDLE mapapi_AddLayer (HANDLE hMap,
HANDLE hGroup,
const char* lpszTitle,
const char* lpszFilename);

Mit der Funktion **mapapi_AddLayer()** können beliebige Layer in eine Karte eingefügt werden. Es kann sich dabei um neue oder leere Layer handeln, oder um Layer die aus einer Datei (lpszFilename) eingelesen werden sollen.

Parameter:

HANDLE *hMap*
Handle einer bestimmten Karte.

HANDLE *hGroup*
Handle der Gruppe, der der Layer angehören soll. Ist dieses Gruppenhandle NULL, so wird der Layer in die aktuell markierte Gruppe der Karte eingefügt. Ist keine Gruppe markiert, so wird er in die letzte Gruppe der Karte eingefügt.

LPSTR *lpszTitle*
Name des Layers. Dieser Name wird in der Baumansicht angezeigt. Ist *lpszTitle* NULL, dann wird standardmäßig der Dateiname ohne Pfad als Layername verwendet.

LPSTR *lpszFilename*
Vollständiger Dateiname eines existierenden Layers. Handelt es sich um einen gültigen Dateinamen, so werden die Daten des Layers aus dieser Datei gelesen. Ist der Dateiname NULL, so wird ein neuer Vektorlayer in der Karte eingefügt, der zu diesem Zeitpunkt noch keine Position enthält.

Es ist möglich sowohl *lpszTitle* als auch *lpszFilename* auf NULL zu setzen. Der Name des Layers ist in diesem Fall die leere Zeichenkette. Er kann nachträglich in der Baumansicht umbenannt werden.

Folgende Dateiformate werden zur Zeit unterstützt:

Dateiendung	Beschreibung
.BMP	Gescannte Karte, die in einem Stück vorliegt (Klasse TBitmapLayer).
.MAP	Karte, die gestückelt vorliegt (Klasse TBmpPieceLayer).
.COR oder .UNC	Internes Format von C&N, beinhaltet GPS-Positionen als ASCII-Text (Klasse TCORLayer).
.DXF	Datei im AutoCAD DXF-Format (Klasse TDXFLayer).

Rückgabewert:
Handle des neuen Layers. Konnte der Layer nicht erzeugt werden, ist der Rückgabewert NULL.

HANDLE mapapi_GetFirstLayer (HANDLE hMap);

Die Funktion liefert ein Handle auf den ersten Layer der Karte unabhängig der Zugehörigkeit zu einer Gruppe.

Parameter:

HANDLE *hMap*
Handle einer bestimmten Karte.

Rückgabewert:
Handle des Layers. Der Rückgabewert ist NULL, wenn die Karte keinen Layer besitzt.

HANDLE mapapi_GetNextLayer (HANDLE hMap);

Die Funktion liefert jeweils das nächste Layer-Handle der Karte unabhängig der Zugehörigkeit zu einer Gruppe.

Parameter:

HANDLE *hMap*
Handle einer bestimmten Karte.

Rückgabewert:
Handle des Layers. Der Rückgabewert ist NULL, wenn die Karte keine weiteren Layer besitzt.

HANDLE mapapi_GetCurSelLayer (HANDLE hMap);

Die Funktion mapapi_GetCurSelLayer() liefert den momentant markierten Layer der Karte.

Parameter:

HANDLE *hMap*
Handle einer bestimmten Karte.

Rückgabewert:
Handle des momentan markierten Layers. Der Rückgabewert ist NULL, wenn kein Layer markiert ist oder die Karte keinen Layer besitzt.

BOOL mapapi_GetLayerData (HANDLE hLayer, LAYER_DATA* pData);

Diese Funktion liefert zusätzliche Daten zum angegebenen Layer.

Parameter:

HANDLE *hLayer*
Handle eines bestimmten Layers.

LAYER_DATA* *pData*
Zeiger auf eine LAYER_DATA Datenstruktur:

```
typedef struct {
  UINT   mask;          // flags that specify which data members are to be used
  LPSTR  pszText;       // name of this layer
  LPSTR  pszFilename;   // data file of the layer
  UINT   type;          // layer type
  HANDLE hGroup;        // handle of group, the layer belongs to
  LPARAM lParam;        // 32bit value associated with the layer
} LAYER_DATA;
```

mask kann eine beliebige Kombination folgender Werte sein:

Gruppenflag	Beschreibung
LYF_TEXT	Der Name des Layers soll zurückgegeben werden. *pszText* muß in diesem Fall einen gültigen Zeiger auf einen Puffer für den Namen enthalten (maximale Länge des Namens: MAX_PATH).
LYF_FILENAME	Der Dateiname des Layers soll ermittelt werden. *pszFilename* muß auf einen genügend großen Puffer (max. Länge eines Dateinames: MAX_PATH) verweisen.
LYF_TYPE	*type* wird gesetzt.
LYF_GROUP	*hGroup* erhält ein Handle auf die Gruppe, der der Layer angehört.
LYF_PARAM	Der *lParam* Wert soll gesetzt werden.

Rückgabewert:

TRUE im Erfolgsfall, FALSE sonst.

BOOL mapapi_SetLayerData (HANDLE hLayer, LAYER_DATA* pData);

Diese Funktion gestattet es, mehrere oder einzelne Eigenschaften des angegebenen Layers zu setzen.

Parameter:

HANDLE *hLayer*

Handle eines bestimmten Layers.

LAYER_DATA* *pData*

Zeiger auf eine LAYER_DATA Datenstruktur.
Näheres siehe dazu **mapapi_GetLayerData()**.

Rückgabewert:

TRUE im Erfolgsfall, FALSE sonst.

BOOL mapapi_AddPos (HANDLE hLayer, GPSAPI_POS* pNewPos);

mapapi_AddPos fügt dem angegebenen Layer am Ende eine neue Position hinzu. Sie wird in die von der Karte (innerhalb der der Layer dargestellt wird) verwendeten Landeskoordinaten umgerechnet und – je nach Layereinstellungen – sofort am Bildschirm dargestellt.

Parameter:

HANDLE *hLayer*

Handle eines bestimmten Layers.

GPSAPI_POS* *pNewPos*

Adresse der neuen Position.

Rückgabewert:
TRUE im Erfolgsfall, FALSE sonst.

---

void mapapi_SetBkColor (HANDLE hMap, COLORREF color);

Setzt die Hintergrundfarbe der angegebenen Karte.

Parameter:

HANDLE *hMap*
Handle der Karte.

COLORREF color
Neue Hintergrundfarbe.

Rückgabewert:
Keiner.

---

void mapapi_GetCurPos (HANDLE hMap,
double* pX, double* pY, double* pZoom);

Ermittelt die aktuelle Position (Mittelpunkt des sichtbaren Kartenausschnittes im Koordinatesystem der Karte) und den aktuellen Zoomfaktor der Karte.

Parameter:

double **x*
Adresse für x-Koordinate

double **y*
Adresse für y-Koordinate

double **zoom*
Adresse für den aktuellen Zoom-Faktor.

Rückgabewert:
Keiner.

---

void mapapi_SetCurPos (HANDLE hMap, double x, double y, double zoom);

Setzte die aktuelle Position (Mittelpunkt des sichtbaren Kartenausschnittes im Koordinatesystem der Karte) und den aktuellen Zoomfaktor der Karte.

Parameter:

double *x*
x-Koordinate

double *y*
y-Koordinate

double *zoom*
Neuer Zoom-Faktor.

Rückgabewert:
Keiner.

**BOOL mapapi_OpenMapDialog (HWND hwndOwner,
LPSTR lpszFilename, int size);**

Öffnet eine Standarddialog zum Öffnen einer Karte. Der Dateiname der Karte wird in *lpszFilename* zurückgegeben.

Parameter:

HWND *hwndOwner*
Fensterhandle der aufrufenden Anwendung.

LPSTR *lpszFilename*
Zeiger auf einen Puffer für den Karten-Dateinamen.

int *size*
Größe des durch lpszFilename adressierten Puffers.

Rückgabewert:
TRUE, wenn der Benutzer eine Karte ausgewählt hat, FALSE sonst.

BOOL mapapi_SaveMap (HANDLE hMap);

Speichert die angegebene Karte unter ihrem aktuellen Dateinamen. Besitzt die Karte noch keinen Dateinamen, wird die Funktion **mapapi_SaveMapAs** aufgerufen.

Parameter:

HANDLE *hMap*
Karten-Handle der zu speichernden Karte.

Rückgabewert:
TRUE, falls die Karte gespeichert wurde, FALSE sonst.

BOOL mapapi_SaveMapAs (HANDLE hMap);

Speichert eine Karte unter einem neuen Dateinamen ab. Die Funktion öffnet den Window-Standarddialog zum Speichern von Dateien. Die Karte wird unter dem vom Benutzer angegebenen Dateienamen abgespeichert.

Parameter:

HANDLE *hMap*
Karten-Handle der zu speichernden Karte.

Rückgabewert:
TRUE, falls die Karte gespeichert wurde, FALSE sonst.

LPSTR mapapi_GetMapFilename (HANDLE hMap);

Liefert einen Zeiger auf den Dateinamen der Karte.

Parameter:

HANDLE *hMap*
Handle einer bestimmten Karte.

Rückgabewert:
Zeiger auf den Dateinamen der Karte. Besitzt die Karte noch keinen Dateinamen, so hat der zurückgelieferte String die Länge 0.

BOOL mapapi_ReadMap (HANDLE hMap, LPSTR lpszFilename);

Liest die Konfiguration einer Karte von der angegebenen Datei.

Parameter:

HANDLE *hMap*
Handle einer bestimmten Karte.

LPSTR *lpszFilename*
Dateiname einer Map-Datei, in dem alle Karteneinstellungen gespeichert sind.

Rückgabewert:
TRUE, wenn die Karte gelesen werden konnte, FALSE sonst.

BOOL mapapi_AddLayerDialog (HANDLE hMap);

Die mapapi_AddLayerDialog-Funkton öffnet den Windows-Standarddialog *GetOpenFile-Name()* zum Auswählen einer Layerdatei. Der Layer wird in die aktuell markierte Gruppe der angegebenen Karte eingefügt.

Parameter:

HANDLE *hMap*
Handle einer Karte.

Rückgabewert:
TRUE, wenn der Layer gelesen und in die Karte eingefügt werden konnte, FALSE sonst.

Dateireferenz

Die unten aufgelisteten Dateien werden zum Compilieren der MAPAPI.DLL benötigt:

Datei	Beschreibung
Verzeichnis *Source:*	
MapAPI.h	Definition aller exportieren DLL-Funktionen
MapAPI.cpp	Implementierung der exportieren Funktionen
MapFrame.h/.cpp	TMapFrameWnd, Rahmenfenster-Objekt
MapWnd.h/.cpp	TMap, Fenster-Objekt für den eigentlichen Kartenausschnitt
LayrTree.h/.cpp	Objekt zur Anzeige der Gruppen und Layer
Symbol.h/.cpp	Objekt zum Zeichnen von Layer-Symbolen
Group.h/.cpp	TGroup
Layer.h/.cpp	TLayer, Basisklasse für alle anderen Layer
VectLayr.h/.cpp	TVectLayer, Basisklasse für alle vektororientierten Layer
CORLayer.h/.cpp	TCORLayer
DXFLayer.h/.cpp	TDXFLayer
BmpLayer.h/.cpp	TBitmapLayer
BmpPcLyr.h/.cpp	TBmpPieceLayer
DLayrOpt.h/.cpp	Layer-Optionen-Dialog
DMapOpt.h/.cpp	Karten-Optionen-Dialog
DMousPos.h/.cpp	Fenster zur Anzeige von Mausposition und Maßstabsleiste
PrevWnd.h/.cpp	Voransichtsfenster
Palette.h/.cpp	Def./Impl. von Funktionen zum Auslesen von Paletteneinträgen aus einer Bitmap-Datei
InfoWnd.h/.cpp	TDistInfoWnd, Dialog zur Anzeige von Distanzen zw.

Verzeichnis *Res:*

MapAPI.rh	Definitionen und Konstanten für alle Resourcen
MapAPI_gr.rc	Beinhaltet alle sprachabhängigen (deutschen) Resourcen (Dialoge, Strings, ...)
Bitmaps.rh/.rc	Bitmaps, Icons, Cursors

Verzeichnis *misc/Source:*

GeoDbase.h/.cpp	TGeoDatabase, Objekt zur Kapselung der DLL-Aufrufe zur Koordinatentransformation
log.h/.cpp	Logfile-Funktionen (für Debugging Zwecke)
LoadDLL.h/.cpp	eigenes TDll-Objekt, kapselt LoadLibrary, FreeLibrary
RegEx.h/.cpp	TRegistry-Objekt, vereinfacht Zugriff auf Registry
coordcvt.h/.cpp	Koordinatenkonvertierungsfunktionen
gcalcex.h/.cpp	Erweiterte Funktionen für den Zugriff auf BlueMarbles Koordinatensystem-Datei (z.B. Löschen von Einträgen)
syseldlg.h/.cpp	Dialog zum Auswählen eines Koordinatensystems
cosysdlg.h/.cpp	Dialog zum Bearbeiten/Definieren von Koordinatensystemen
datumdlg.h/.cpp	Dialog zum Bearbeiten/Definieren eines geodätischen Datums
ellipdlg.h/.cpp	Dialog zum Bearbeiten/Definieren von Ellipsoiden
geocalc.h/.cpp	„GeoCalculator", Koordinaten-Transformationsdialog
geodg_gr.rh/.rc	Resourcen der Koordinatendialoge
Textfile.h/cpp	Objekt zum zeilenweisen Einlesen von Textdateien
getnames.h/.cpp	Funktionen zum Extrahieren von Dateinamen aus einer von *GetOpenFilename()* zurückgegebenen Dateiliste.
corpfile.h/.cpp	Objekt zum Auslesen von .cor- und .unc-Dateien (C&N eigenes Dateiformat)

Kapitel 6
Benutzerhandbuch

Behandelten alle bisherigen Kaptiel die programmtechnischen Aspekte, so widmet sich nun dieses Kapitel der Konfiguration und der Anwendung des Softwarepaketes. Es soll hier aber nur ein grober Überblick gegeben werden, bei spezielle Fragen sei auf die Online-Hilfe verwiesen.

Installation und Deinstallation

Installation

Um das Programm zu starten ist keine gesonderte Installationsvorgang notwendig. Es läßt sich ohneweiteres auch von der CD aufrufen (GeoLocator.exe).

Es wird in diesem Fall jedoch nicht das ‚Data'-Unterverzeichnis als Arbeitsverzeichnis verwendet, sondern das temporäre Verzeichnis des lokalen Arbeitsplatzrechners. Sollen also bereits vorhandene Dateien (z.B. Fahrzeugverwaltung, Adreßbuch, etc...) verwendet werden, so sind diese vom ‚Data'-Verzeichnis der CD in das temporäre Verzeichnis zu kopieren und das *ReadOnly*-Bit zurückzusetzen. Das gleiche gilt auch dann, wenn das Programm von einem Netzwerklaufwerk gestartet wird, und der Benutzer keine Schreibrechte auf das Data-Verzeichnis besitzt.

Wird das Programm zum ersten Mal gestartet, so sind – sofern vorhanden – Gps-Gerätetreiber über das Menü Extras/GPS-Setup einzurichten.

Deinstallation

Zur Deinstallation ist erstens das Programmverzeichnis, in das das System kopiert wurde zu löschen und zweitens die Einträge in der Registry zu entfernen. Folgende drei Einträge können gelöscht werden:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Communication & Navigation\GeoLocator
HKEY_LOCAL_MACHINE\SOFTWARE\Communication & Navigation\GPSAPI
HKEY_LOCAL_MACHINE\SOFTWARE\Communication & Navigation\MAPAPI
```

Die letzten beiden Einträge sollten jedoch nur dann gelöscht werden, wenn außer dem GeoLocator keine andere Anwendung das GPSAPI oder das MAPAPI verwendet.

Konfiguration

Wenn das Programm zum ersten mal gestartet wird und ein GPS-Gerät angeschlossen ist muß ein Gerätetreiber eingerichtet werden. Öffnen Sie dazu über den Menüpunkt Extras/Gps-Setup den Dialog GPS-Setup. Mit dem Befehl hinzufügen öffnet sich ein Dialog, indem sie den gewünschten Gerätetyp auswählen, und die Einstellungen für die serielle Schnittstelle vornehmen können.

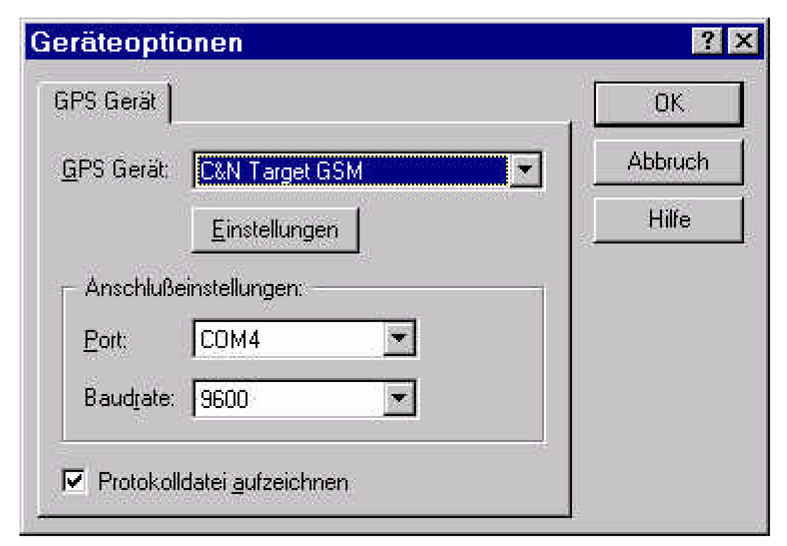


Abb. 31. Der Geräteoptionen-Dialog

Vergessen beim Anschluß eines GSM-Gerätes nicht auf die Eingabe Ihrer PIN! Diese ist Dialog „GSM Optionen" einzugeben. Sie erreichen diesen Dialog mit dem Befehl „Einstellungen" im oben abgebildeten Dialog:

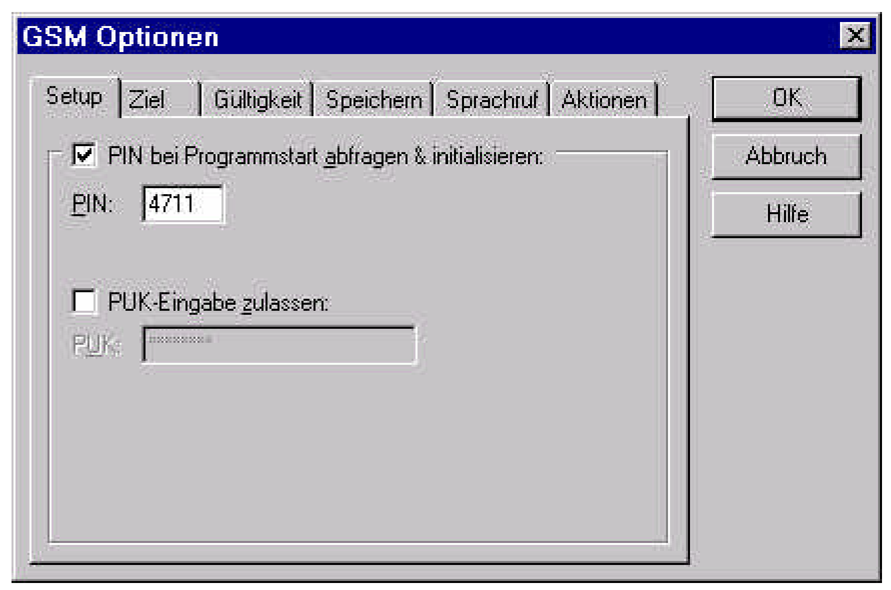


Abb. 32. Der GSM-Optionen-Dialog

Falls Sie die PIN falsch angegeben haben ist auch die Eingabe der PUK notwendig. Schließen alle Setup-Dialog mit OK. Im Ereignisfenster des GeoLocators erhalten Sie entsprechende Mitteilungen vom GPS-Gerätetreiber, ob Ihre Empfänger, bzw. GSM-Modul richtig konfigurert wurden.

Bedienung

Dieser Abschnitt gibt einen Überblick über die Befehle des Hauptmenüs.

Menüeintrag	***Beschreibung***
Datei	
Österreich 1:500.000	*Öffnet die vordefinierte, mitgelieferte Österreich-Karte*
Neue Karte	*Öffnet eine neue, leere Karte*
Öffnen	*Erlaubt das Laden einer eigenen Karte*
Speichern	*Speichert die Einstellungen der aktuellen Karte*
Speichern unter ...	*Speichert die aktuelle Karte unter einem neuen Namen*
Schließen	*Schließt das aktive MDI-Fenster*
Beenden	*Beendet das Programm*
Bearbeiten	
Rückgängig	*Letzte Änderungen rückgängig machen*
Ausschneiden	*Löscht und kopiert Nachrichten in die Zwischenablage (nur verfügbar, wenn das Ereignisfenster geöffnet ist)*
Kopieren	*Kopiert eine Nachrichtin die Zwischenablage (nur im Ereignisfenster verfügbar)*
Einfügen	*(dzt. noch nicht verfügbar)*
Löschen	*Löscht entweder den ausgewählten Layer von der Karte entweder Ereignisse aus dem Ereignsfenster*
Layer einfügen	*Fügt der aktuellen Karte neue Layer hinzu.*
Ansicht	
Fahrzeugverwaltung	*Öffnet den Dialog für die Fahrzeugverwaltung*
GPS Gerätestatus	*Öffnet wahlweise für jedes angeschlossenes GPS-Gerät ein nicht-modales Fenster, das laufend die aktuellen Daten anzeigt.*
Ereignisanzeige	*Öffnet ein Fenster zur Darstellung von Ereignissen, Textnachrichten und Fahrzeugpositionen*
Layerliste	*Blendet in der aktiven Karte die Layerliste ein, bzw. aus*
Maßstabsleiste	*Blendet in der aktiven Karte die Maßstabsleiste am unteren Rand ein, bzw. aus*
Extras	
Neue Textnachricht	*Öffnet einen Dialog zum Verfassen einer neuen Textnachricht (nur verfügbar wenn das Ereignisfenster aktiv)*
Textnachricht beantw.	*Erlaubt das Beantworten der aktuell markierten Textnachricht*
Telefonbuch	*Öffnet den Telefonbuch-Dialog*
Optionen	*Öffnet einen Dialog für allgemeine Programmeinstellungen*
Koordinaten	*Behfehle zum Einstellen von Koordinatensystemen:*
Koordinatensystem auswählen	*Wählt für die aktuelle Karte ein bestimtes Koordinatensystem aus*
Koordinatentransformation	*Öffnet einen Dialog zur Umrechnung von Koordinaten*
Koordinatensystem definieren	*Erlaubt das Erstellen und Bearbeiten von Koordinatensystemen*

Datum definieren	*Öffnet den Dialog zum Bearbeiten eines geodätischen Datums*
Ellipsoid definieren	*Öffnet den Dialog zum Bearbeiten von Ellipsoiden*
GPS-Setup	*Öffnet den GPS-Setup Dialog*
Fenster	*Allg. Fensterbefehle (Windows-Standard)*
Hilfe	
Info über ...	*Zeigt Versionsnummer und Copyright-Rechte an.*

Anhang

NAVSTAR GPS, Acronyme und Abkürzungen

AE	-	Antenna Electronics
A/D	-	Analog to Digital
AFB	-	Air Force Base
AFI	-	Automatic Fault Indication
AFS	-	Air Force Station
AHRS	-	Attitude and Heading Reference System
AIMS	-	Airspace Traffic Control Radar Beacon
A/J	-	Anti-Jamming
AOC	-	Auxilliary Output Chip
A-S	-	Anti-Spoofing
ASIC	-	Application Specific Integrated Circuit
ATE	-	Automatic Test Equipment
BCD	-	Binary Code Decimal
BIH	-	Bureau International de L'Heure
BIPM	-	International Bureau of Weights and Measures
BIT	-	Built-In-Test
BPSK	-	Bi Phase Shift Keying
C/A-code	-	Coarse/Acquisition-Code
CADC	-	Central Air Data Computer
CDMA	-	Code Division Multiplex Access
CDU	-	Control Display Unit
CEP	-	Circular Error Probable
CMOS	-	Complementary Metal Oxide Semiconductor
C/No	-	Carrier to Noise Ratio
CRPA	-	Controlled Radiation Pattern Antenna
CSOC	-	Consolidated Space Operations Center
CW	-	Continuous Wave
DAC	-	Digital to Analog Converter
dB	-	Decibel (X = 10 Log X dB)
DGPS	-	Differential GPS
D-Level	-	Depot Level
DLM	-	Data Loader Module
DLR	-	Data Loader Receptable
DLS	-	Data Loader System
DMA	-	Defense Mapping Agency
DoD	-	Department of Defense
DOP	-	Dilution of Precision
dRMS	-	Distance Root Mean Square
DRS	-	Dead Reckoning System
DT&E	-	Development Test and Evaluation
ECEF	-	Earth-Centered-Earth-Fixed
ECP	-	Engineering Change Proposal
EDM	-	Electronic Distance Measurement
EFIS	-	Electronic Flight Instrument System
EM	-	Electro Magnetic
EMCON	-	Emission Control
ESGN	-	Electrically Suspended Gyro Navigator
FAA	-	Federal Aviation Administration
FMS	-	Foreign Military Sales
FOM	-	Figure Of Merit
FRPA	-	Fixed Radiation Pattern Antenna
FRPA-GP	-	FRPA Ground Plane

GaAs	-	Gallium Arsenide
GDOP	-	Geometric Dilution of Precision
GMT	-	Greenwich Mean Time
GPS	-	Global Positioning System
HDOP	-	Horizontal Dilution of Precision
HOW	-	Hand Over Word
HSI	-	Horizontal Situation Indicator
HV	-	Host Vehicle
HQ USAF	-	Headquarters US Air Force
ICD	-	Interface Control Document
ICS	-	Initial Control System
IF	-	Intermediate Frequency
IFF	-	Identification Friend or Foe
I-Level	-	Intermediate Level
ILS	-	Instrument Landing System
INS	-	Inertial Navigation System
ION	-	Institute of Navigation
IOT&E	-	Initial Operational Test and Evaluation
IP	-	Instrumentation Port
ITS	-	Intermediate Level Test Set
JPO	-	Joint Program Office
J/S	-	Jamming to Signal Ration
JTIDS	-	Joint Tactical Information Distribution System
L1	-	GPS primary frequency, 1575.42 MHz
L2	-	GPS secondary frequency, 1227.6 MHz
LEP	-	Linear Error Probable
LRIP	-	Low Rate Initial Production
LRU	-	Line Replaceable Unit
LO	-	Local Oscillator
mB	-	Millibar
MCS	-	Master Control Station
MCT	-	Mean Corrective Maintenance Time
MLV	-	Medium Launch Vehicle
MmaxCT	-	Maximum Corrective Maintenance Time
MOU	-	Memorandum of Understanding
M/S	-	Metres per Second
MSL	-	Mean Sea Level
MTBF	-	Mean Time Between Failure
MTBM	-	Mean Time Between Maintenance
N/A	-	Not Applicable
NAV-msg	-	Navigation Message
NOSC	-	Naval Ocean Systems Center
NRL	-	Naval Research Labratory
NS	-	Nanosecond
NSA	-	National Security Agency
NTDS	-	Navy Tactical Data System
NTS	-	Navigation Technology Satellite
OBS	-	Omni Bearing Select
OCS	-	Operational Control System
O-Level	-	Organization Level
OTHT	-	Over The Horizon Targeting
PC	-	Personal Computer
P-Code	-	Precise Code
PDOP	-	Position Dilution of Precision
PLSS	-	Precision Location Strike System
P I	-	Pre Planned Product Improvement
PPM	-	Parts Per Million
PPS	-	Precise Positioning Service
PPS-SM	-	PPS Security Module
PRN	-	Pseudo Random Noise
PTTI	-	Precise Time and Time Interval
PVT	-	Position Velocity and Time
RAM	-	Reliability and Maintainability

RCVR	-	Receiver
RF	-	Radio Frequency
RMS	-	Root Mean Square
RNAV	-	Area Navigation
RSS	-	Root Sum Square
RT	-	Remote Terminal
RTCA	-	Radio Technical Commission for Aeronautics
RTCM	-	Ratio Technical Commission for Maritime Services
S/A	-	Selective Availability
SAMSO	-	Space and Missile Systems Organization
SBB	-	Smart Buffer Box
SC	-	Special Committee
SEP	-	Spherical Error Probable
SI	-	International System of Units
SIL	-	System Integration Labratory
SINS	-	Shipborne INS
SPS	-	Standard Positioning Service
SRU	-	Shop Replacable Unit
STDCDU	-	Standard CDU
TACAN	-	Tactical Air Navigation
TAI	-	International Atomic Time
TBD	-	To Be Determined
TDOP	-	Time Dilution of Precision
TFOM	-	Time Figure Of Merit
TTFF	-	Time to First Fix
UE	-	User Equipment
UERE	-	User Equivalent Range Error
UHF	-	Ultra High Frequency
USA	-	United States of America
USNO	-	US Naval Observatory
UT	-	Universal Time
UTC	-	Universal Time Coordinated
VDOP	-	Vertical Dilution of Precision
VHSIC	-	Very High Speed Integrated Circuit
VLSIC	-	Very Large Scale Integrated Circuit
VOR	-	Very High Frequency (VHF) Omnidirectional Range
WGS-84	-	World Geodetic System – 1984
YPG	-	Yuma Proving Ground
1 PPM	-	1 Pulse Per Minute
1 PPS	-	1 Pulse Per Second

Literaturverzeichnis

[ANGER96] ANGERMANN, D.; BAUSLERT, G.; KLOTZ, J.; REINKIG, J.; ZHU, S.Y.: Hochgenaue Koordinatenbestimmung in großräumigen GPS-Netzen. In: Allgemeine Vermessungsnachricht. 103. Jg., Heft 5, S.185-195. Verlag Herbert Wichmann, Heidelberg, 1996.

[BARW96] BARWINSKI, K. et al.: Einmessung von Erdgas-Hochdruckleitungen mit GPS - wirtschaftlich eine Alternative? In: Allgemeine Vermessungsnachricht. 103. Jg., Heft 6, S.196-202. Verlag Herbert Wichmann, Heidelberg, 1996.

[BILL94] BILL, R.; FRITSCH, D.: Grundlagen der Geo-Informationssysteme. Band 1: Hardware, Software und Daten. 2. Aufl. Herbert Wichmann Verlag, Heidelberg, 1994.

[EBER97] Eberspächer, Jörg, GSM, Global System for Mobile Communication: Vermittlung, Dienste und Protokolle in digitalen Mobilfunktnetzen; mit 41 Tabellen / von Jörg Eberspächer und Hans-Jörg Vögel. – Teubner, Stuttgart 1997, ISBN 3-519-06192-9

[FÜRST96] FÜRST, P.: Vermessung mit GPS und Laser-Feldstecher im Wald. In: Salzburger geographische Materialien, Angewandte geographische Informationsverarbeitung VIII, Heft 24, S.120-125. Hrsg. Strobel, Dollinger, Salzburg, 1996.

[GSM0102] ETSI GSM 01.02: Digital cellular telecommunications system (Phase 2+); General description of a GSM Public Land Mobile Network (PLMN)

[GSM0201] ETSI GSM 02.01: Digital cellular telecommunications system (Phase 2+); Principles of telecommunication services supported by a GSM Public Land Mobile Network (PLMN)

[GSM0202] ETSI GSM 02.02: Digital cellular telecommunications system (Phase 2+); Bearer Services (BS) supported by a GSM Public Land Mobile Network (PLMN).

[GSM0203] ETSI GSM 02.03: Digital cellular telecommunications system (Phase 2+); Teleservices supported by a GSM Public Land Mobile Network (PLMN).

[GSM0302] ETSI GSM 03.02: Digital cellular telecommunications system (Phase 2+); Network architecture.

[GSM0338] ETSI GSM 03.38: Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information.

[GSM0340] ETSI GSM 03.40: Digital cellular telecommunications system (Phase 2+); Technical realization of the Short Message Service (SMS) Point-to-Point (PP)

[GSM0705] ETSI GSM 07.05: Digital cellular telecommunications system (Phase 2+); Use of Data Terminal Equipment - Data Circuit terminating; Equipment

(DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS)

[GSM0707] ETSI GSM 07.07: European digital cellular telecommunication system (Phase 2); AT command set for GSM Mobile Equipment.

[GSM1114] ETSI GSM 11.14: Digital cellular telecommunications system (Phase 2+); Specification of the SIM application toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface

[HOKL94] HOFMANN-WELLENHOF, B.; KIENAST, G.; LICHTENEGGER, H.: GPS in der Praxis. Springer Verlag, Wien, New York, 1994.

[HOLC94] HOFMANN-WELLENHOF, B.; LICHTENEGGER, H.; COLLINS, J.: Global Positioning System, Theory and Practice, S. 353. Springer Verlag, Wien, New York, 1994.

[HUBER96] HUBER, K.: Echtzeit-DGPS für alle? In: Zeitschrift für Vermessungswesen, 121. Jg., Heft 9, S. 455-460. Verlag Konrad Wittwer, Stuttgart, 1996.

[MOTO94] Motorola Oncore User's Guide, For Use with ONCORE Receiver Board Firmware Release 4.X, 5.X and 6.X, Motorola, 1994

[PUNDT96] PUNDT, H.; BRINKKÖTTER-RUNDE, K.; STREIT, U.: GPS-unterstützte digitale Felddatenerfassung für Geoinformationssysteme in Land- und Forstwirtschaft. In: Salzburger geographische Materialien, Angewandte geographische Informationsverarbeitung VIII, Heft 24, S. 110-119. Hrsg. Strobel, Dollinger, Salzburg, 1996.

[SCHRÖ94] Schrödter, Frank: GPS-Satelliten-Navigation. Technik, Systeme, Geräte, Funktionen und praktischer Einsatz; Franzis, Poing 1994, ISBN 3-7723-6682-1

[SAUER93] SAUERMANN, K.: GPS-Verfahren für den Nahbereich mit kurzen Beobachtungszeiten in Vermessung und Ortung. In: Deutsche Geodätische Kommission bei der Bayrischen Akademie der Wissenschaften, Heft 403, Reihe C. Dissertation, Fachbereich Vermessungswesen der Technischen Hochschule Darmstadt, 1993.

[STRO95] Strobel, Jürgen: Global Positioning System: GPS; Technik und Anwendung der Satellitennavigation. Franzis, Poing 1995, ISBN 3-7723-4202-7

[UNAVCO] UNAVCO University NAVSTAR Consortium http://www.unavco.ucar.edu

[WANN96] WANNINGER, L.: Präzise GPS-Positionierungen in regionalen Netzen permanenter Referenzstationen. In: Zeitschrift für Vermessungswesen, 121. Jg., Heft 9, S. 441-454. Verlag Konrad Wittwer, Stuttgart 1996.

[WITTE95] WITTE, B.; SCHMIDT, H.: Vermessungskunde und Grundlagen der Statik für das Bauwesen. 3. neubearb. Auflage. Verlag Konrad Wittwer, Stuttgart 1995.

[WOOD85] WOODEN, W., H.: Navstar Global Positioning System. In: Proceedings of the First International Symposion on Precise Positioning with the GPS, vol 1: p.23-32. Rochville, Maryland, April 15-19th, 1985.